IEC 61375-2-1

Edition 1.0 2012-06

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Electronic railway equipment – Train communication network (TCN) – Part 2-1: Wire Train Bus (WTB)

Matériel électronique ferroviaire – Réseau embarqué de train (TCN) – Partie 2-1: Bus de Train Filaire (WTB)

IEC 61375-2-1:2012

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC

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

About IEC publications

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

Useful links:

IEC publications search - www.iec.ch/searchpub

The advanced search enables you to find IEC publications by a variety of criteria (reference number, text, technical committee,…).
It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available on-line and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary (IEV) on-line.

Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de la CEI

La Commission Electrotechnique Internationale (CEI) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications CEI

Le contenu technique des publications de la CEI est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Liens utiles:

Recherche de publications CEI - www.iec.ch/searchpub

La recherche avancée vous permet de trouver des publications CEI en utilisant différents critères (numéro de référence, texte, comité d'études,…).
Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

Just Published CEI - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications de la CEI. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (VEI) en ligne.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

IEC 61375-2-1

Edition 1.0 2012-06

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Electronic railway equipment – Train communication network (TCN) – Part 2-1: Wire Train Bus (WTB)

Matériel électronique ferroviaire – Réseau embarqué de train (TCN) – Partie 2-1: Bus de Train Filaire (WTB)

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE
CODE PRIX XP

ICS 45.060

ISBN 978-2-88912-067-3

Warning! Make sure that you obtained this publication from an authorized distributor.
Attention! Veuillez vous assurer que vous avez obtenu cette publication via un distributeur agréé.

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

ELECTRONIC RAILWAY EQUIPMENT – TRAIN COMMUNICATION NETWORK (TCN) –

Part 2-1: Wire Train Bus (WTB)

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 61375-2-1 was prepared by IEC technical committee 9: Electrical equipment and systems for railways.

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

FDIS	Report on voting
9/1642/FDIS	9/1666/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 2.

A list of all parts of IEC 61375 series, under the general title *Electronic railway equipment – Train communication network (TCN)*, can be found on the IEC website.

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

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

This first edition cancels and replaces the clauses of IEC 61375-1 second edition published in 2007, relevant to the specification of WTB and constitutes a technical revision.

It was prepared taking into account IEC 61375-1, third edition.

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

INTRODUCTION

This part of IEC 61375 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 consists which are frequently coupled and uncoupled, as is the case of international UIC trains.

Figure 1 illustrates the WTB application.

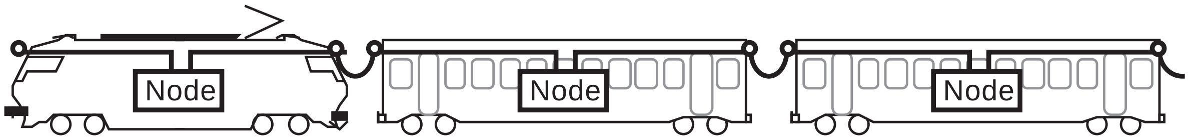


Figure 1 – Wire Train Bus

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 networks, a Train Backbone and a Consist network:

a) for interconnecting consists 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 a Consist network e.g. the Multifunction Vehicle Bus (MVB) can be used.

In the TCN architecture, WTB features Real-Time Protocols, which offer two communication services:

c) Process Variables, a distributed, real-time database, periodically refreshed through broadcasting;

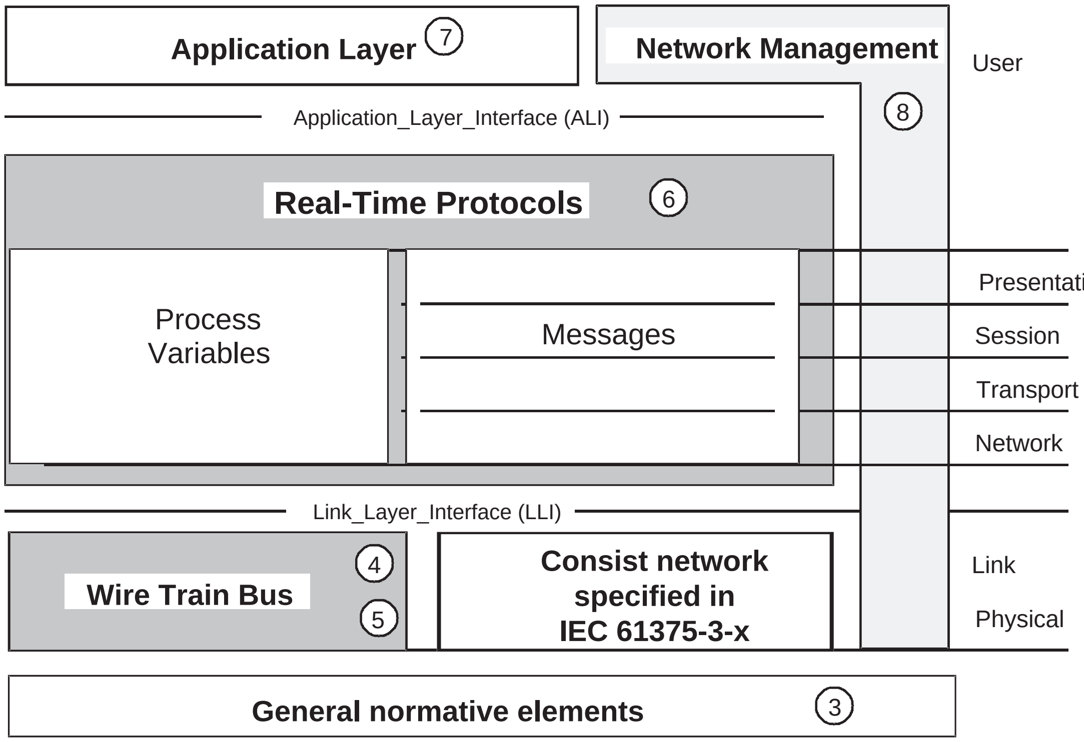
d) messages, transmitted on demand either as:

- unicast messages (point-to-point) or/and
- multicast messages.

WTB in the TCN offers a common Network Management, which allows debugging, commissioning and maintenance over the network.

The Consist network MVB shares Real-Time Protocols and Network Management with WTB. Other implementations of consist networks need adaption to the Real-Time Protocols and Network Management of WTB.

The TCN is structured similarly to the Open System Interconnection model defined in ISO/IEC 7498-1 (see Figure 2).



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

Figure 2 – Layering of the TCN

This standard has been, for editorial reasons, divided into eight clauses:

Clause 1

– Scope;

Clause 2

– Normative references;

Clause 3

– Terms and definitions, abbreviations, conventions;

Clause 4 and 5: Wire Train Bus,

– Physical layer and Link Layer Control;

Clause 6: Real-Time protocols,

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

Clause 7: Application Layer

– Process Data Marshalling
– WTB Line Fault Location Detection

Clause 8: Train Network Management

– Configuration, supervision and control of the network.

ELECTRONIC RAILWAY EQUIPMENT – TRAIN COMMUNICATION NETWORK (TCN)–

Part 2-1: Wire Train Bus (WTB)

1 Scope

This part of IEC 61375 applies to data communication in Open Trains, i.e. it covers data communication between consists of the said open trains and data communication within the consists of the said open trains.

The applicability of this standard to the train communication bus (WTB) allows for interoperability of individual consists within Open Trains in international traffic. The data communication bus inside consists (e.g. MVB) is given as recommended solution to cope with the said TCN. In any case, proof of compatibility between WTB and a proposed consist network 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 Clause 3.

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

2 Normative references

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

IEC 60571, *Electronic equipment used on rail vehicles*

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

IEC 61375-1, *Electronic railway equipment – Train communication network (TCN) – Part 1: General architecture*

IEC 61375-2-2:2012, *Electronic railway equipment – Train communication network (TCN) – Part 2-2: Wire Train Bus conformance testing*

IEC 61375-3-1, *Electronic railway equipment – Train communication network (TCN) – Part 3-1: Multifunction Vehicle Bus (MVB)*

ISO/IEC 8802-2, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 2: Logical link control*

ISO/IEC 8824 (all parts), *Information technology – Abstract Syntax Notation One (ASN.1)*

ISO/IEC 8825 (all parts), *Information technology – ASN.1 encoding rules*

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

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

ISO/IEC 10646, *Information Technology – Universal Multipl-Octet Coded Character Set (UCS)*

ISO/IEC 13239, *Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures*

ITU-T Recommendation V24*, List of definitions for interchange circuits between data terminal equipment (DTE) and data-circuit terminating equipment (DCE)*

ITU-T Recommendation Z.100, *Specification and Description Language (SDL)*

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

UIC CODE 556, *Information transmission in the train (train-bus)*

UIC CODE 557, *Diagnostics on passenger rolling stock*

3 Terms and definitions, abbreviations, conventions

3.1 Terms and definitions

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

3.1.1
address
identifier of a communication partner, of which several types exist, depending on the layer

3.1.2
agent
application process in a Station which accesses the local managed objects on behalf of the Manager

3.1.3
Aperiodic Data
transmission of Process Data on a demand basis. This service is not used

3.1.4
Application Layer
upper layer in the OSI model, interfacing directly to the Application

3.1.5
Application Layer Interface
definition of the services offered by the Application Layer

3.1.6
Application Messages Adapter
code directly called by the application implementing the Messages services

3.1.7
Application Messages Interface
definition of the Messages services

3.1.8
Application Process
communicating entity, implemented for instance by a task

3.1.9
Application Processor
processor which runs a communicating Application Process

3.1.10
Application Supervision Interface
definition of the Supervision services available in particular to the Agent

3.1.11
Application Variables Adapter
code directly called by the application implementing the Variables services

3.1.12
Application Variables Interface
definition of the Variables services

3.1.13
arbiter
device, or common protocol followed by several devices, which selects one of several devices competing for mastership

3.1.14
Auxiliary Channel
channel used for detecting additional Nodes

3.1.15
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)

3.1.16
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

3.1.17
bit-stuffing
method specified by ISO/IEC 13239 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

3.1.18
bridge
device which stores and forwards frames from one bus to another on the base of their Link Layer addresses

3.1.19
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

3.1.20
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

3.1.21
Bus Controller
processor or integrated circuit in charge of the Link Layer of communication

3.1.22
Bus Switch
switch or relay within a WTB Node which connects electrically the cable sections of the two directions

3.1.23
Caller
Application Process which initialises a message exchange

3.1.24
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

3.1.25
Check Variable
Process Variable of type antivalent boolean protecting another Process Variable

3.1.26
Check Offset
bit offset of a Check Variable within a Dataset

3.1.27
Closed Train
train consisting of a set of consists, where the composition does not change during normal operation, for instance metro, suburban train, or high-speed train units

3.1.28
composition
number and characteristics of the consists forming a train

3.1.29
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

3.1.30
Connect Confirm
response of the Consumer to the Connect Request of the Producer

3.1.31
Connect Request
first packet of a message sent from Producer to Consumer

3.1.32
Consist
Singe vehicle or a group of vehicles which are not separated during normal operation. A Consist could contain no, one or several Consist networks.

3.1.33
Consist network
bus connecting equipment within a consist, e.g. the MVB, and which conforms or adapts to the TCN Real-Time protocols as described in this document

3.1.34
consistency
Dataset consisting of several elements is consistent if all elements are read or written in one indivisible operation

3.1.35
Consumer
receiver of a message at the Transport Layer (see: Producer)

3.1.36
continuity consist
consist without an operational Train Bus Node, but carrying a section of the bus to connect passively the Train Bus of its adjacent consists.

3.1.37
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

3.1.38
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

3.1.39
Dataset
all Process Variables transmitted in one Process Data frame

3.1.40
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

3.1.41
Destination Device
receiver of a frame at the Link Layer (see: Source Device)

3.1.42
device
unit connected to one or more busses

3.1.43
Device Address
Device Address identifies a device within a bus; 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

3.1.44
Direction 1
one direction of a WTB Node

3.1.45
Direction 2
other direction of a WTB Node

3.1.46
End Delimiter
sequence which ends a frame before the medium returns to idle

3.1.47
End Node
Node which terminates the two bus segments connected to it but does not establish continuity between them

3.1.48
Event Round
sequence of polls in which all events pending at the start are read

3.1.49
extension box
wiring box where the trunk cable is interrupted and passively extended by an extension cable to connect a device

3.1.50
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

3.1.51
field device
device attaching simple sensors and actuators to the bus, outside a rack

3.1.52
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)

3.1.53
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 13239

3.1.54
frame
sequence of consecutive symbols sent in one time slot by a transmitter, between two slots where the line is idle

3.1.55
Frame Check Sequence
16-bit FCS specified in ISO/IEC 13239

3.1.56
Frame Data
data transmitted between the Preamble and the End Delimiter (on the WTB)

3.1.57
fritting
electrical cleaning of oxidised contacts by applying a breakdown voltage over the contact

3.1.58
Function
Application Process which exchanges messages with another Application Process

3.1.59
Function Directory
directory which maps a Function Identifier to a Station Identifier and vice-versa

3.1.60
Function Identifier
8-bit identifier of a Function

3.1.61
F code
in a Master Frame, indicates the request and the expected response Slave Frame size

3.1.62
gateway
connection between different busses at the Application Layer requiring application-dependent data analysis and protocol conversion

3.1.63
Group Address
address of a multicast group to which a Node belongs

3.1.64
Group Directory
directory which indicates to a Node to which multicast group it pertains

3.1.65
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

3.1.66
HDLC
High-level Data Link Control, a set of standardised protocols, including ISO/IEC 13239 for data transmission

3.1.67
HDLC Data
data transmitted in an HDLC frame

3.1.68
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

3.1.69
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

3.1.70
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)

3.1.71
integrity
property of a system to recognise and to reject wrong data in case of malfunction of its parts

3.1.72
Intermediate Node
Node which establishes continuity between two bus sections connected to it, but does not terminate them

3.1.73
jumper cable
cable connecting the trunk cables of two consecutive consists, 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 consists

3.1.74
Line
non-redundant bus. A dual-thread bus consists of two lines

3.1.75
Line Unit
all circuits providing the electrical attachment to a line

3.1.76
Link Address
address supplied to the Link Layer to identify to which Bus and to which Device Address a packet is sent or received

3.1.77
Link Control
field in the HDLC frame which indicates the type of frame

3.1.78
Link Data
data transported by the Link Layer, but not relevant to it

3.1.79
Link Header
part of a Message Data frame relevant to the Link Layer

3.1.80
Link Layer
layer in the OSI model establishing point-to-point and broadcast connections between devices attached to the same bus

3.1.81
Link Layer Interface
interface between Link Layer and higher communication layers

3.1.82
Link Layer Management
interface controlling the Link Layer for management purposes

3.1.83
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

3.1.84
local area network
part of a network characterised by a common medium access and address space

3.1.85
logical link control
protocols and associated frame formats which serve to control the Link Layer

3.1.86
Logical Address
address which is not bound to a specific device (e.g. the Process Data address)

3.1.87
Logical Port
ports of a device used for the Process Data traffic and addressed by the Logical Address

3.1.88
Macro Cycle
number of Basic Periods corresponding to a Macro Period

3.1.89
Macro Period
longest Individual Period, after which the periodic traffic returns to the same pattern, counted in milliseconds

3.1.90
Main Channel
channel over which the main bus traffic is received

3.1.91
Management Message
message exchanged between a Manager and an Agent for Network Management.

3.1.92
Manager
Function in a Station which is dedicated to Network Management and which send management Call Messages through System Addresses

3.1.93
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

3.1.94
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

3.1.95
Master Frame
frame sent by a Master

3.1.96
Master Start Delimiter
Start Delimiter of a Master Frame

3.1.97
medium access control
sublayer of the Link Layer, which controls the access to the medium (arbitration, mastership transfer, polling)

3.1.98
medium dependent interface
mechanical and electrical interface between the transmission medium and a Medium Attachment Unit

3.1.99
medium
physical carrier of the signal: electrical wires, optical fibres, etc.

3.1.100
Medium Attachment Unit
device used as a coupler to the transmission medium

3.1.101
message
data item transmitted in one or several packets

3.1.102
Messages
transmission service of the TCN

3.1.103
Message Data
data transmitted sporadically by the Link Layer in relation to message transmission; the corresponding Link Layer service

3.1.104
messenger
communication stack caring for end-to-end message communication and interfacing to the application

3.1.105
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

3.1.106
Multifunction Vehicle Bus
MVB
Consist network to be used for connecting programmable stations and simple sensors/actors.

3.1.107
Multiple Unit Train
train consisting of a set of closed trains, where the composition of the set may change during normal operation

3.1.108
network
set of possibly different communication systems which interchange information in a commonly agreed way

3.1.109
Network Address
address which identifies a Function or a Station within the network. It can be either a User Address or a System Address

3.1.110
Network Header
part of a Message Data frame relevant to the Network Layer

3.1.111
Network Layer
layer in the OSI model responsible for routing between different busses

3.1.112
Network Management
operations necessary to remotely configure, monitor, diagnose and maintain the network

3.1.113
Node
device on the Wire Train Bus, which may act as a gateway between Train Bus and Consist network

3.1.114
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

3.1.115
Node Descriptor
24-bit data structure which indicates for a Node its Node Period and its Node Key

3.1.116
Node Directory
directory which maps the Node Address to the Device Address (one-to-one mapping in WTB)

3.1.117
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

3.1.118
Node Period
on the WTB, desired Individual Period of a Node (identical to Individual Period except if overload occurs)

3.1.119
octet
8-bit word stored in memory or transmitted as a unit *

3.1.120
Open Train
train consisting of a set of consists where the configuration may change during normal operation, for instance international UIC trains

3.1.121
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)

3.1.122
packet
unit of a message (information, acknowledgement or control) transmitted in exactly one Message Data frame

3.1.123
period
time unit after which a periodic pattern repeats itself

3.1.124
Periodic Data
Process Data transmitted periodically, at an interval which is the Individual Period

3.1.125
Periodic List
list of Nodes, addresses or devices to be polled in each period of a Macro Cycle

3.1.126
Periodic Phase
phase during which the Master polls for Periodic Data according to its Periodic List

3.1.127
Physical Address
The Node Address on the WTB which identify communicating devices on the same bus

---

* IEC prescribes 'octet ' instead of 'byte'.

3.1.128
Physical Port
Port used for the Message Data or the Supervisory Data traffic and addressed by the Device Address

3.1.129
pitch
distance between adjacent devices on the same electrical bus required to avoid clustering of bus loads

3.1.130
polling
sending of a Master Frame in order to receive a Slave Frame

3.1.131
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)

3.1.132
Port Index Table
look-up table which deduces the memory address of a port from the Logical Address of the Process Data

3.1.133
Preamble
sequence of signals heading a frame for the purpose of synchronising the receiver, used on the WTB

3.1.134
Presentation Layer
layer in the OSI model responsible for data representation and conversion

3.1.135
Process Data
source-addressed data broadcast periodically by the link layer in relation with Process Variables transmission; the corresponding Link Layer service

3.1.136
Process Variable
variable expressing the state of a process (e.g. speed, brake command)

3.1.137
Producer
sender of a message at the Transport Layer (see: Consumer)

3.1.138
Publisher
source of a Dataset for broadcasting (see: Subscriber)

3.1.139
PV Name
identifier of a Process Variable

3.1.140
PV Set
set of Process Variables belonging to the same Dataset

3.1.141
queue
memory storing an ordered set of frames in a first-in, first-out fashion

3.1.142
rack
equipment containing one or more devices, attached to the same segment

3.1.143
reassembly
act of regenerating a long message from several packets generated by segmentation

3.1.144
receiver
electronic device which may receive signals from the physical medium

3.1.145
Receive Queue
queue for receiving Message Data in a device

3.1.146
regular operation
normal bus activity as opposed to Inauguration (WTB)

3.1.147
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

3.1.148
Replier
Application Process which has been requested by the Caller to receive a Call Message and to reply with a Reply Message

3.1.149
residual error rate
probability of integrity breach (unrecognised wrong bit) per transmitted bit

3.1.150
router
connection between two busses at the Network Layer, which forwards datagrams from one bus to another on the base of their Network Address

3.1.151
scan
polling of devices in a certain sequence for supervisory purposes

3.1.152
section
part of a segment, which is passively connected to another section without terminator in between

3.1.153
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

3.1.154
segmentation
division of a long message into several shorter frames for transmission

3.1.155
Send Queue
queue for sending Message Data in a device

3.1.156
service
capabilities and features of a sub-system (e.g. a communication layer) provided to a user

3.1.157
Session Header
part of a Message Data frame relevant to the Session Layer

3.1.158
Session Layer
OSI layer in charge of establishing and closing communication

3.1.159
Side A
one side of a consist with respect to a WTB Node

3.1.160
Side B
other side of a consist with respect to a WTB Node

3.1.161
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

3.1.162
Slave Frame
frame sent by a Slave

3.1.163
Source Device
sender of a frame at the Link Layer (see: destination device)

3.1.164
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)

3.1.165
Sporadic Data
data frames transmitted on demand to carry Message Data or Supervisory Data

3.1.166
Sporadic Phase
second half of a Basic Period, dedicated to the demand-driven transmission of messages and bus management data

3.1.167
star coupler
device which takes the light of an optical fibre and redistributes it to several other fibres

3.1.168
Station
device capable of message communication, by contrast to simple devices, and which supports an Agent Function

3.1.169
Station Directory
directory which maps a Station Identifier to a Link Address and vice-versa

3.1.170
Station Identifier
8-bit identifier of a Station

3.1.171
Station Status Word
16-bit descriptor of the status and capabilities of a Station

3.1.172
Strong Master
Strong Node is currently Master and will not relinquish mastership until demoted to Weak Node status

3.1.173
Strong Node
Node selected by the application to become Strong Master. There may be only one Strong Master on a bus segment

3.1.174
stub
T-connection branching from an electrical bus line (at the tap), connecting a device to the line

3.1.175
Subscriber
one of the sinks of a broadcast Dataset (see: Publisher)

3.1.176
Supervisory Data
data transmitted within one bus only for the purpose of Link Layer supervision (e.g. Inauguration on the WTB)

3.1.177
System Address
Network Address of a Management Message exchanged between Manager and Agent, consisting of Node Address and Station Identifier

3.1.178
tap
place where a segment is tapped. A tap is a three-way electrical fork

3.1.179
Telegram
Master Frame and the corresponding Slave Frame, treated as a whole

3.1.180
terminator
circuit which closes an electrical transmission line, ideally by its characteristic impedance

3.1.181
Terminator Switch
switch which inserts the Terminator at the end of a segment on the WTB

3.1.182
Topography
data structure describing the Nodes attached to the Train Bus, including their address, orientation, position and Node Descriptor

3.1.183
topology
possible cable interconnection and number of devices a given network supports

3.1.184
Topography Counter
counter in a Node which is incremented at each new Inauguration

3.1.185
Traffic Store
shared memory accessed both by the network and the user, which contains the Process Data Port

3.1.186
Train Communication Network
data communication network for connecting programmable electronic equipment on-board rail vehicles

3.1.187
Train Bus, Train Backbone
bus connecting the consists of a train, in particular, the WTB, and which conforms to the TCN protocols

3.1.188
Train Network Management
services of the Network Management for TCN

3.1.189
transceiver
combination of a transmitter and of a receiver

3.1.190
transmitter
electronic device which can transmit a signal on the physical medium

3.1.191
Transport Data
data carried by the Transport Layer, but not relevant to it

3.1.192
Transport Header
part of a Message Data frame relevant to the Transport Layer

3.1.193
Transport Layer
layer of the OSI model responsible for end-to-end flow control and error recovery

3.1.194
trunk cable
cable which runs along the consists, as opposed to extension cable or jumper cable

3.1.195
User Address
Network Address of a User Message exchanged between Functions, consisting of Node Address (or Group Address) and Function Identifier

3.1.196
User Message
messages exchanged between user Functions

3.1.197
Variables
transmission service of the TCN

3.1.198
Var_Offset
bit offset of a Process Variable within a Dataset

3.1.199
Vehicle Descriptor
application-dependent information about a particular vehicle, such as length and weight

3.1.200
Weak Master
Weak Node is currently Master and which will relinquish mastership if it finds another, stronger Master

3.1.201
Weak Node
Node which may take over the bus mastership spontaneously, but which releases it if it detects a stronger Node

3.1.202
Wire Train Bus (WTB)
Train Bus for frequently coupled and uncoupled consists, such as international UIC trains

3.2 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 μs
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
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 13239
HDLC	High-level Data Link Control, a Link Layer protocol whose frame format is defined in ISO/IEC 13239
IEC	International Electrotechnical Commission, Geneva
IEEE	Institute of Electrical and Electronics Engineers, New York
ISO	International Standard Organisation, Geneva
LFLD	Line Fault Location Detection
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 Consist network
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
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
PDM	Process Data Marshalling
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, issued by UIC
RTP	Real-Time Protocols, the common communication protocols given in Clause 6 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 consist networks and Train Backbones
TNM	Train Network Management
UIC	International Union of Railways , the international railways operators association
WTB	Wire Train Bus

3.3 Conventions

3.3.1 Base of numeric values

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

Analog and fractional values include a comma.

EXAMPLE 1 The voltage is 20,0 V.

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

EXAMPLE 2 Decimal 20 coded on 8 bits = ‘0001 0100’B = ‘14’H.

3.3.2 Naming conventions

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

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

When a data structure is associated with a keyword, its type consists of the same basic words 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 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.

3.3.3 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.

3.3.4 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 1 lm_send_request

Constants and type definitions appear in all upper case.

EXAMPLE 2 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	*The service or data type is expressed here.* *In case of an indication procedure, the event which triggers the call is indicated here, beginning with "When"* *The name and parameters of the service procedure are defined here.* *In case of an indication procedure, the type of the procedure is specified.* *Input parameters, output parameters and return parameters are distinguished*	

Syntax	`MD_RESULT lm_send_request`
	`( /* example */`
	`unsigned, destination,`
	`UNSIGNED8 link_control,`
	`MD_PACKET * p_packet`
	`ENUM8 * status`
	`)`	

Input	*Input parameters are supplied to the procedure, which is not allowed to modify them*	
	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	*Output parameters are expected to be modified by the call*	
	status	The type ENUM8 is an 8-bit enumeration type
Result	*The Result parameter is an optional output parameter which expresses success or failure of the call, but not necessarily of the service*	
	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 3.3.5.		

3.3.5 Specification of transmitted data

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

a) a graphical form, which is not normative, but shows the message structure at a glance;

b) a textual form based on ASN.1, with encoding rules specified in 6.3.

EXAMPLE 1 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->	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	snu	gni	node_id						station_id							
2	next_station_id								rsv1	Tvd	topo_counter					
4	tnm_key								sif_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															
	parameter5.3: STRING32															
	(CHARACTER8)								last character or 0							
^octet																

The bit numbering follows the representation of power of two value in a byte or word. The bit numbering does not indicate the sequence of transmission on the bus, which can be different: MSB first or LSB first.

In the graphical form, one line is used for each word of 16 bits, but in Clause 5 (WTB Link Layer Control), 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 2).

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

3.3.6 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 3 shows an example of a state transition diagram.

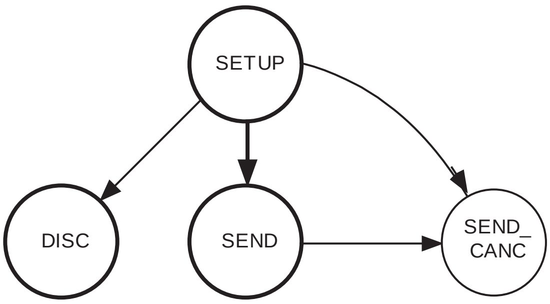


Figure 3 – State transition example

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

The transition between these states is 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:

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

3.4 General considerations

3.4.1 Interface between equipment

This standard defines the data communication interface of equipment located in a consist as a connection of devices to a Consist network, as shown in Figure 4.

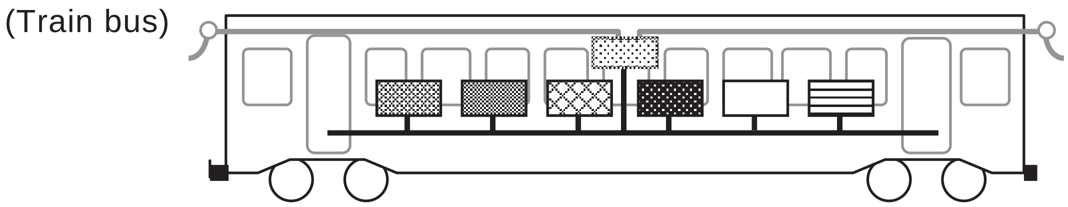


Figure 4 – Interfaces between equipment

A Consist Network is designed primarily, but not exclusively, for interconnecting equipment where interoperability and interchangeability is needed.

3.4.2 Interface between consists

This standard defines the data communication interface between consists as the connection of Nodes located in consists to a Train Bus, as shown in Figure 5.

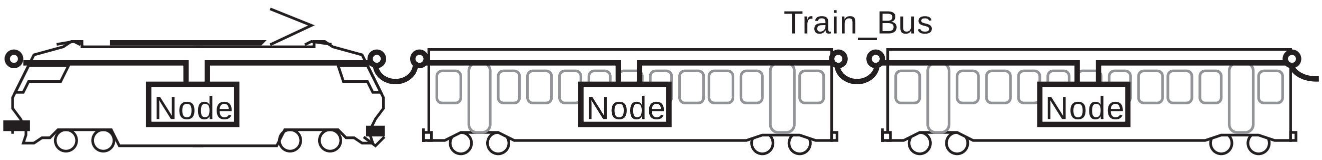


Figure 5 – Interfaces between consists

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

NOTE For a definition of Open Trains, see Clause 3.

3.4.3 Real-Time Protocols

This standard defines the architecture of the TCN as a hierarchy consisting of two levels, a Train Bus and a Consist network, as shown in Figure 6.

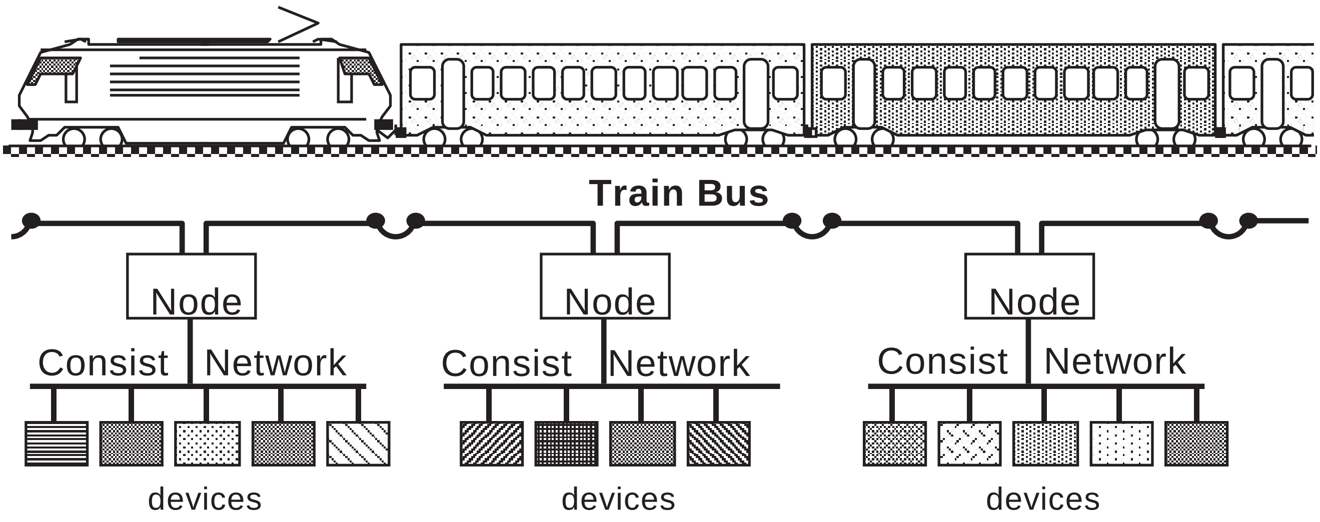


Figure 6 – Train Bus and Consist network

As communication protocols, Clause 6 of this standard specifies the Real-Time Protocols (RTP), used by all nodes on the WTB. The devices on the Consist network can use the same RTP (e.g. MVB) or adapt the Consist network protocol to the RTP of the WTB nodes.

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.

3.4.4 Network Management

As Network Management, Clause 8 of this standard specifies the TCN Network Management (TNM) as a collection of messages exchanged between Manager and Agent to provide the basic services.

3.4.5 Configurations

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

a) the WTB without a consist network or with another consist network than MVB;
b) the RTP with other busses than the WTB or the MVB.

Figure 7 shows three configurations corresponding to three application domains:

c) 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 consist. To each Node, up to 15 Consist networks (MVB or others) may be attached;
d) 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 Consist network can span several vehicles;

e) the Closed Trains configuration shows a closed train, in which the Consist network (e.g. MVB) can be used both as Train Bus and as Consist network.

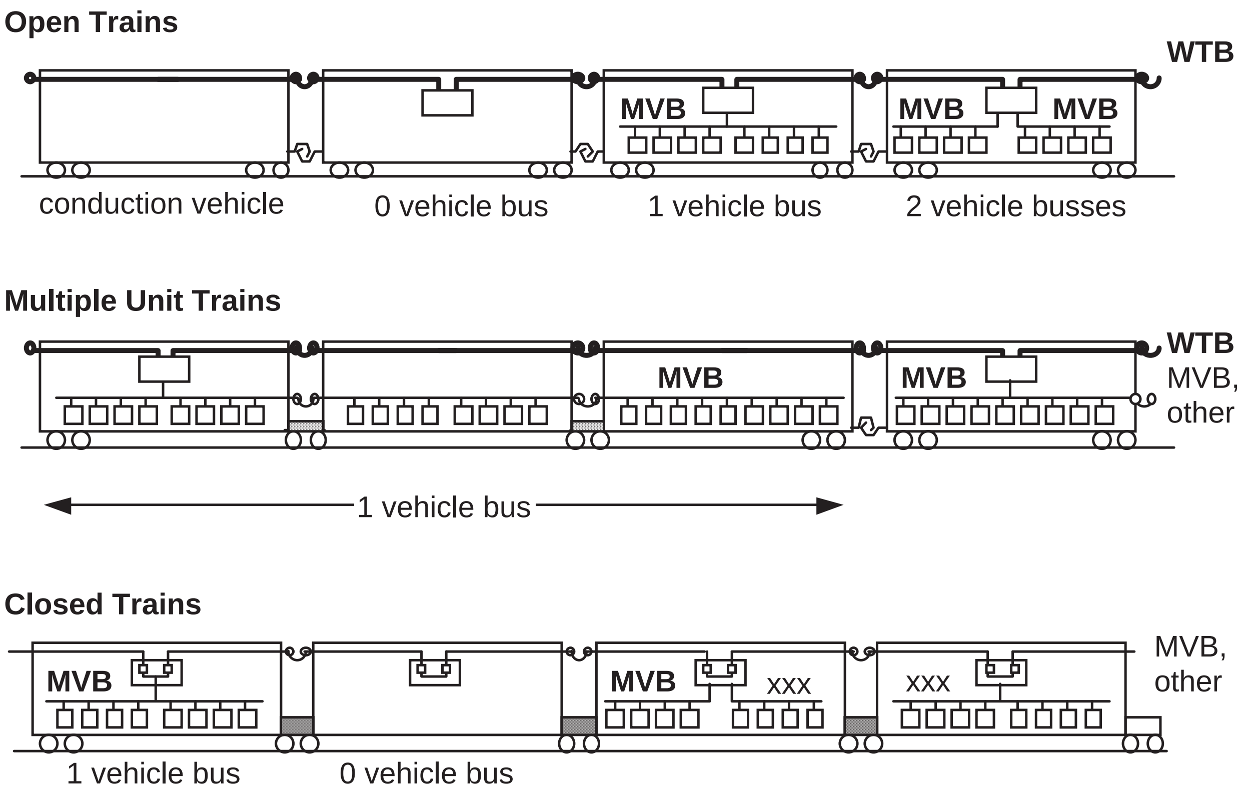


Figure 7 – TCN configurations

In all three configurations of Figure 7 the Consist network MVB is used to connect on-board equipment, but other busses may be used as Consist network.

3.4.6 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 8.

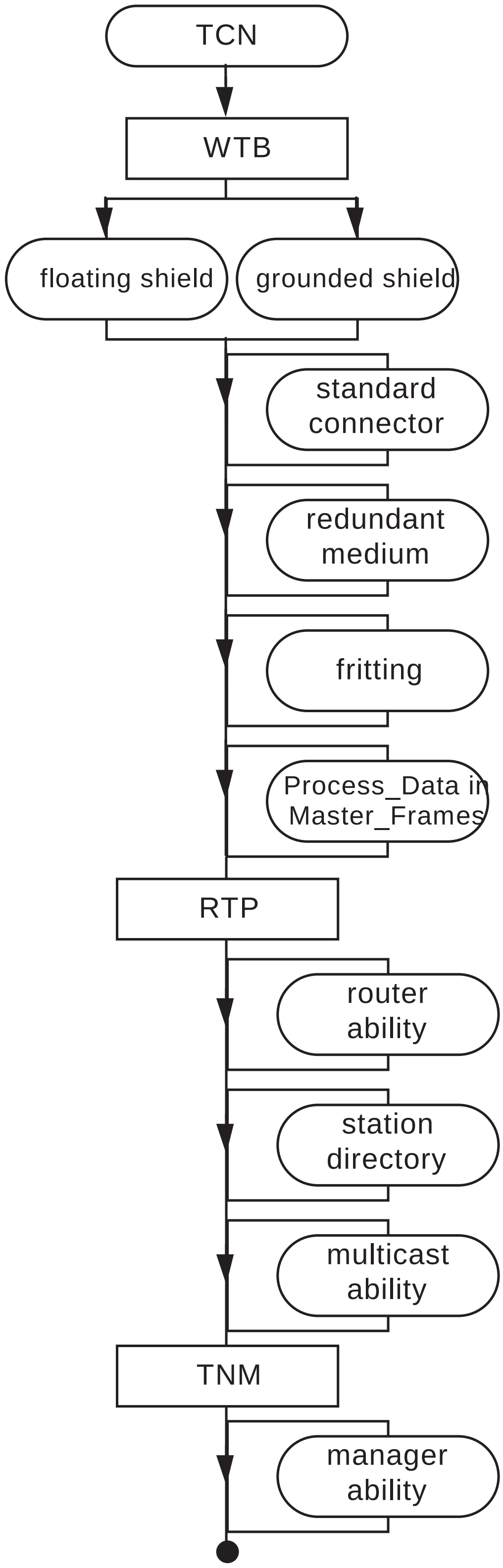


Figure 8 – TCN WTB device configuration options

3.4.6.1 TCN device

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

3.4.6.2 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 fritting as specified in this standard.

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

3.4.6.3 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.

3.4.6.4 TNM options

A TCN device shall implement the Agent function.

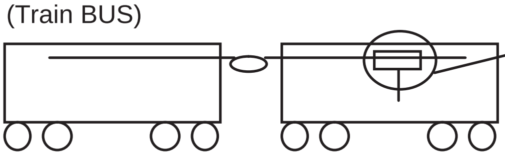
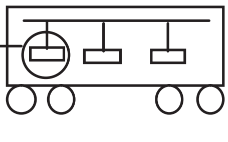
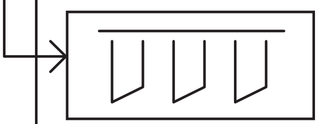
A TCN device may implement the Manager function.

3.5 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 IEC 61375-2-2.

Table 5 – Interoperability testing

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

4 Physical layer

This clause 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.

4.1 Topology

4.1.1 Bus sections

The WTB bus shall consist of nodes interconnected by bus sections of the following types:

a) trunk cables running along a consist (continuity vehicles only have a trunk cable);

b) jumper cables connecting the trunk cables of different consists;

c) extension cables extending the trunk cable to reach a node.

4.1.2 Couplers

Connectors and junction boxes may be used to assemble nodes and cable sections.

Each consist carries a portion of the bus and a number of nodes.

4.1.3 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 node per consist or several nodes per one single vehicle in a train, as shown in Figure 9.

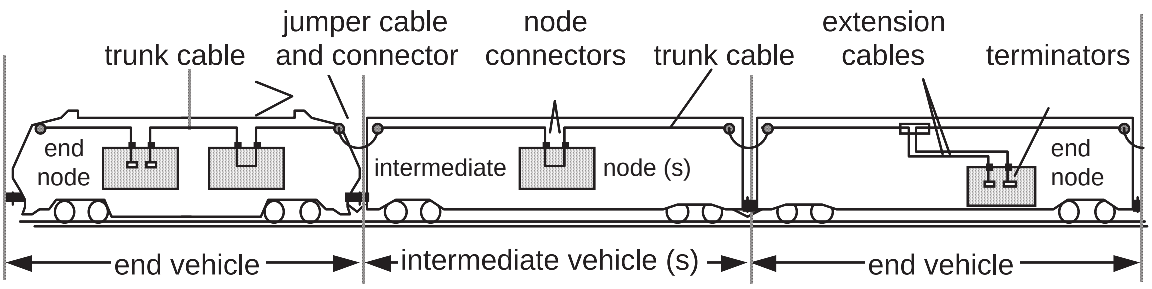


Figure 9 – Train Composition (two Intermediate Nodes shown)

4.1.4 Consist orientation

Where orientation of the nodes is critical to the left-right recognition, the following conventions shall be observed:

a) one end of the consist is identified as Extremity 1, the other as Extremity 2;

c) if Direction_1 points north, the side of the consist 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 consist 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 consist, since the orientation of consists with respect to one another is not predictable.

NOTE 2 Extremity 1 of a consist is the end of the consist opposite to where the parking brake is located.

4.1.5 Consist specification (informal)

Since a manufacturer may supply individual consists or nodes rather than a complete train, the specifications of the bus, the consist and the nodes are treated separately.

This standard specifies the characteristics of the whole bus and of each node. For a particular application, the consist characteristics are deduced from them. The conformance of the train is then met providing the number of vehicles, consists or nodes is not exceeded.

The following calculations apply to railway vehicles as specified in UIC CODE 556. Other applications, such as mass transit, may use a similar calculation.

The reference train composition in UIC CODE 556 consists of 22 vehicles, giving a cable length of 860,0 m without the use of repeaters. There is normally only one node per consist, but up to 10 consists composed of one single vehicle (e.g. driving trailer) may support a second node, for a total of 32 nodes.

According to 4.5.2, 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 × 0,3 dB = 9,6 dB.

According to 4.6.3, 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 dB.

The maximum attenuation allocated to a vehicle is therefore 10,4 dB/22 = 0,5 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 10.

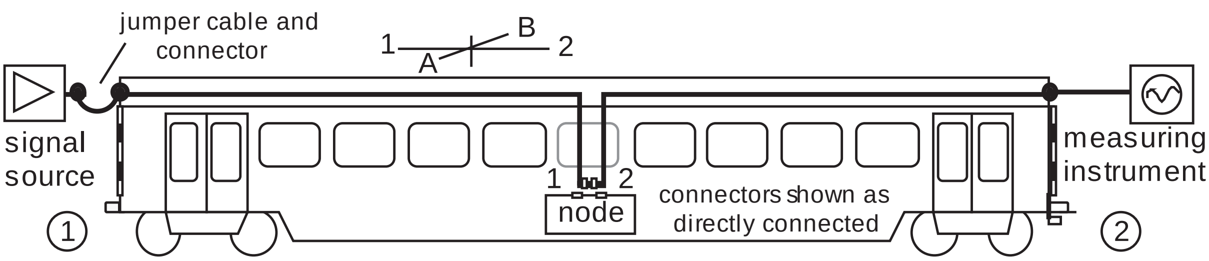


Figure 10 – Vehicle measurement

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 × 26,0 × 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 (4.2.4).

The same principle is applicable to the other distortion parameters. The measurement scheme is explained in 4.5.2.1.

When the consist is equipped with redundant Line_A and Line_B, the test applies to each line individually.

4.2 Medium specifications

4.2.1 Topology

Nodes shall be inserted into the WTB cable, each node being attached to two bus sections, as shown in Figure 11.

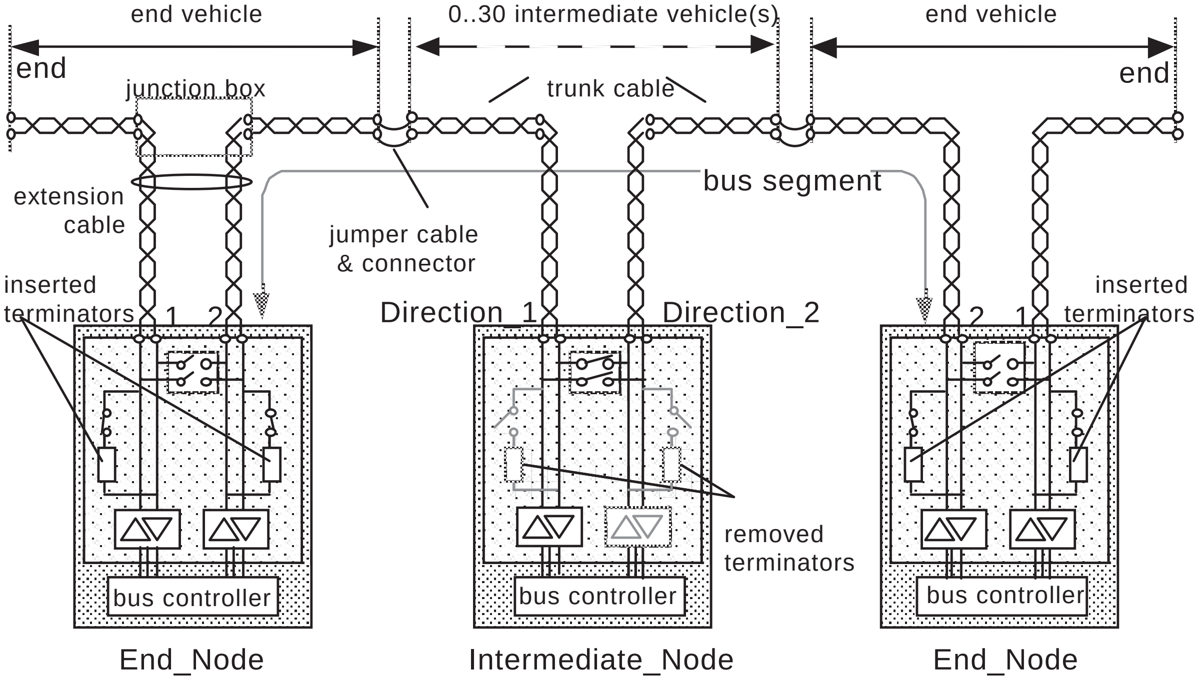


Figure 11 – Connected nodes in regular operation

A node shall be able either:

a) to establish electrical continuity between the two bus sections connected to it, to perform as Intermediate Node, or

b) 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 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 12.

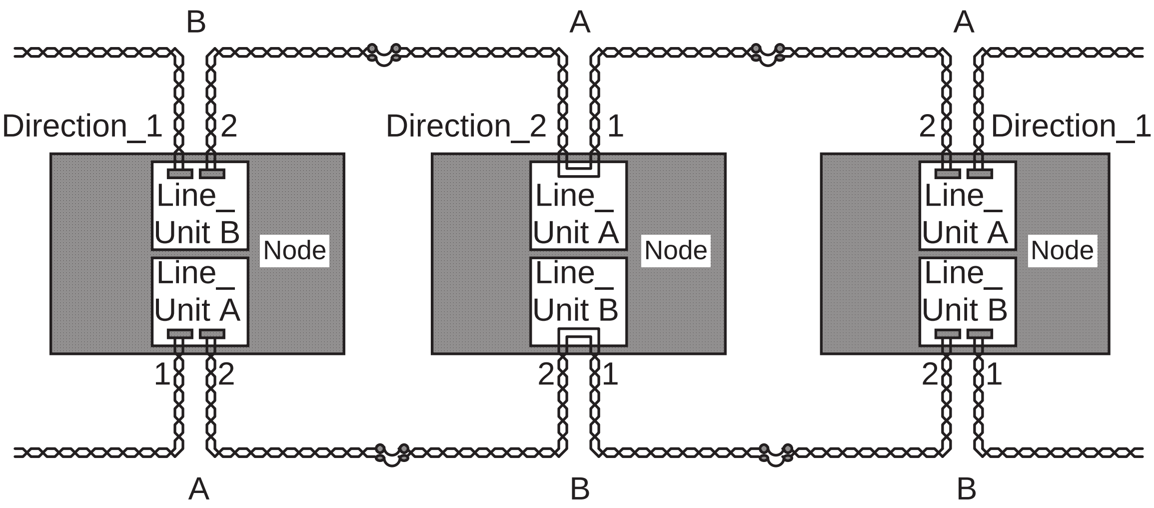


Figure 12 – Double-line attachment

In case this option is used, the following specifications apply:

a) the lines shall be identified as Line_A and Line_B;

b) this identification shall be consistent for all nodes within the same consist;

c) cables belonging to different lines shall be marked distinctly;

d) 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 consist, and Line_B associated with side B.

NOTE 3 Since the orientation of consists is not predictable, Line_A of one consist may be connected to Line_B of another consist, as shown in Figure 12.

4.2.3 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 ± 0,01 % sinusoidal signal with a differential amplitude of ± 4,0 V (8,0 Vpp).

4.2.3.1 Signalling speed

All bus segments shall operate at the same speed of 1,0 Mbit/s ± 0,01 %, which, due to the Manchester encoding, corresponds to a signalling frequency of 1,0 MHz (BT = 1,0 μs, BR = 1,0 MHz).

4.2.3.2 Delay due to nodes and cabling

T_pd, the end-to-end propagation delay on the bus shall not exceed 60,0 μs between any two nodes.

NOTE 1 The propagation delay for a given application may be evaluated as

T_pd = (L × 6,0 + R × T_rd)

where

L is the length of the cable (trunk, extension and jumper cables) in m;

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 5.2.2.2 and 4.7.2.2.

4.2.3.3 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 4.6.3.

4.2.3.4 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 ± 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 Vpp ± 10 %, centred on 0,0 V, through a source impedance of 22,0 Ω ± 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 4.6.3.

4.2.3.5 Skew between redundant lines (option)

The maximum difference in propagation delay between Line_A and Line_B shall not exceed T_skew = 30,0 μs between any two nodes.

NOTE This specification matches the receiver's tolerable skew as specified in 4.7.2.4.1.

4.2.4 Cable specification

4.2.4.1 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 mm^2 (AWG 18).

The recommended cross-section of the jumper cable wires is 1,34 mm^2 (AWG 16).

If indirect attachment through Sub-D connectors as in 4.3.4 is used, the cross-section of each wire of the extension cable shall be no more than 0,56 mm^2 (AWG 20).

4.2.4.2 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.

4.2.4.3 Characteristic impedance

All bus sections shall present a differential characteristic impedance of Zw = 120,0 Ω (±10 %) measured with a sinusoidal signal at a frequency between 0,5 BR and 2,0 BR.

4.2.4.4 Cable attenuation

It is recommended that 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.

4.2.4.5 Distributed capacitance

The differential (wire-to-wire) distributed capacitance of the cable shall not exceed 65 pF/m at 1,0 BR.

4.2.4.6 Capacitive unbalance to shield

The capacitive unbalance to shield shall not exceed 1,5 pF/m at 1,0 BR.

4.2.4.7 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.

4.2.4.8 Shield quality

The transfer impedance of the cable, shall, at 20,0 MHz, be less than 20,0 mΩ/m, measured as specified in 5.1.5.1.2. of IEC 61375-2-2.

The differential transfer impedance of the cable shall be less than 2,0 mΩ/m, measured as specified in 5.1.5.1.2 of IEC 61375-2-2.

4.2.4.9 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 then 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 following the method specified in 5.1.5.1.2 of IEC 61375-2-2.

4.2.5 Shielding concept

To satisfy different applications, two shielding concepts are specified:

- a grounded shield concept (preferred method) and
- a floating shield concept.

4.2.5.1 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 13.

NOTE 1 The shields should be connected to ground whenever possible, for example 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.

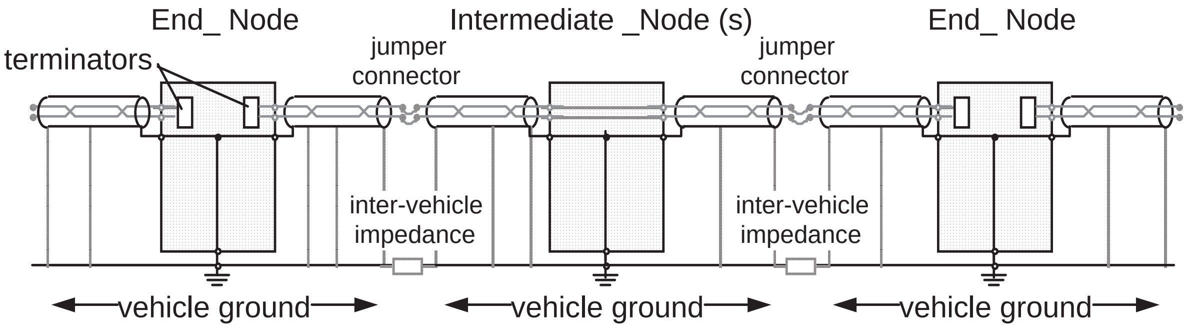


Figure 13 – Grounded shield concept

NOTE 2 The grounded shield concept is specified by the UIC 558 leaflet.

4.2.5.2 Floating shield concept

When applying the floating shield concept:

the shield 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 Rs = 47,0 kΩ ± 5 % in parallel with a capacitor Cs = 100,0 nF ± 10 %, 750,0 V, as shown in Figure 14:

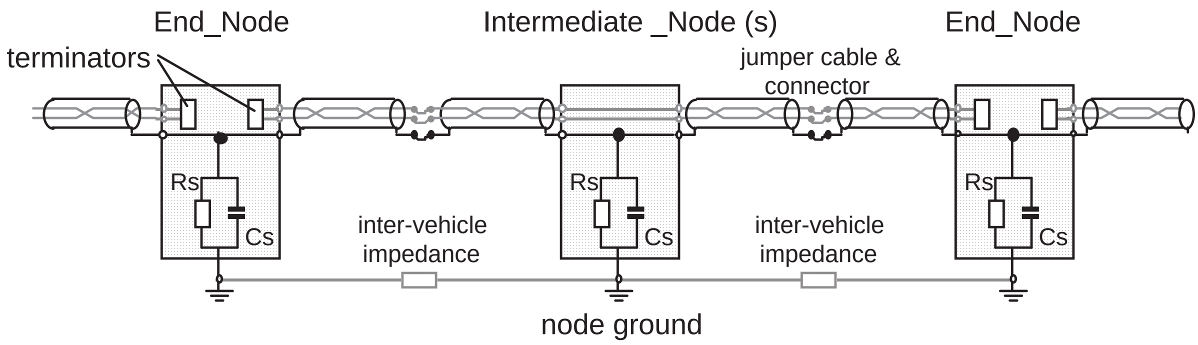


Figure 14 – Floating shield concept

4.2.6 Terminator

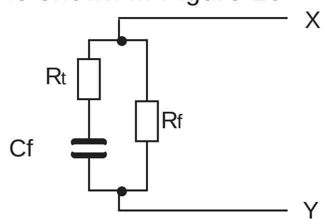
The End Nodes shall electrically terminate the two bus segments to which they are connected with a terminator.

The terminator shall be a non-polarised resistor with a value of Zw ± 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 kΩ 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 15.



Components	Type	Value
Rt	resistor	130,0 Ω
Rf	resistor	2,4 kΩ
Cf	capacitor	47,0 nF

Figure 15 – Terminator

4.3 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.

4.3.1 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 and 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).

4.3.2 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 16:

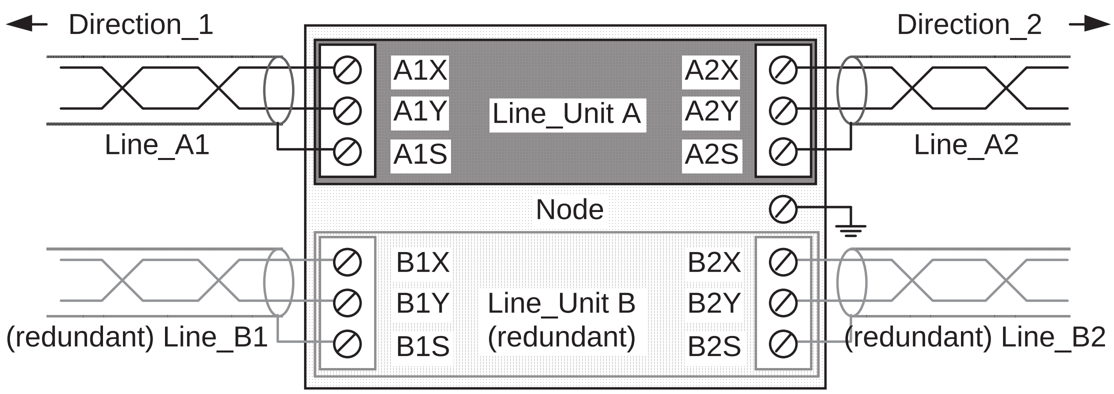


Figure 16 – Direct node attachment (optional double-line)

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.

4.3.3 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 17.

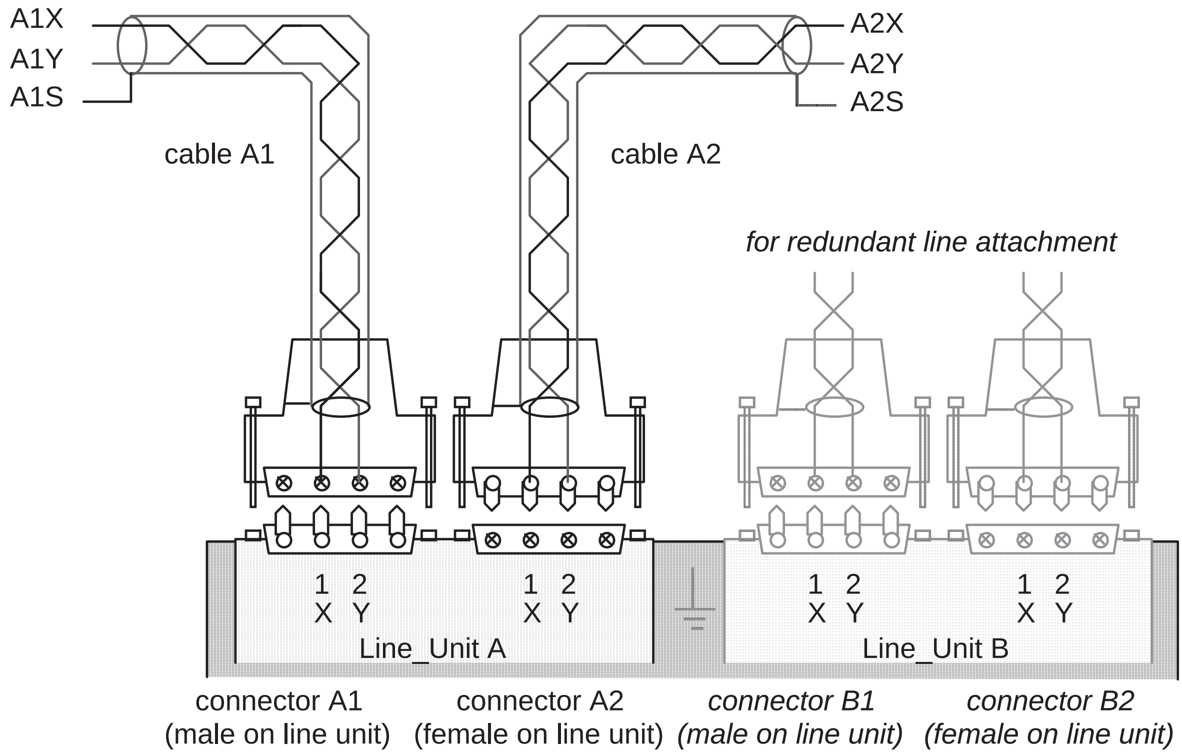


Figure 17 – Indirect attachment

4.3.4 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, so 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 6, as shown in Figure 18.

Table 6 – 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		

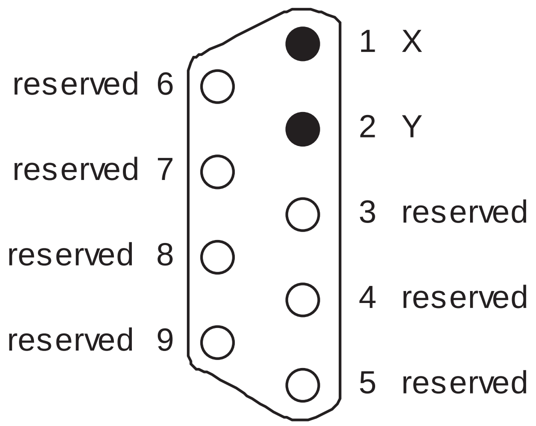


Figure 18 – WTB connector, front view

NOTE It is preferable for the floating shield concept to electrically separate the receptacle from the node case.

4.4 Node specifications

4.4.1 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 19.

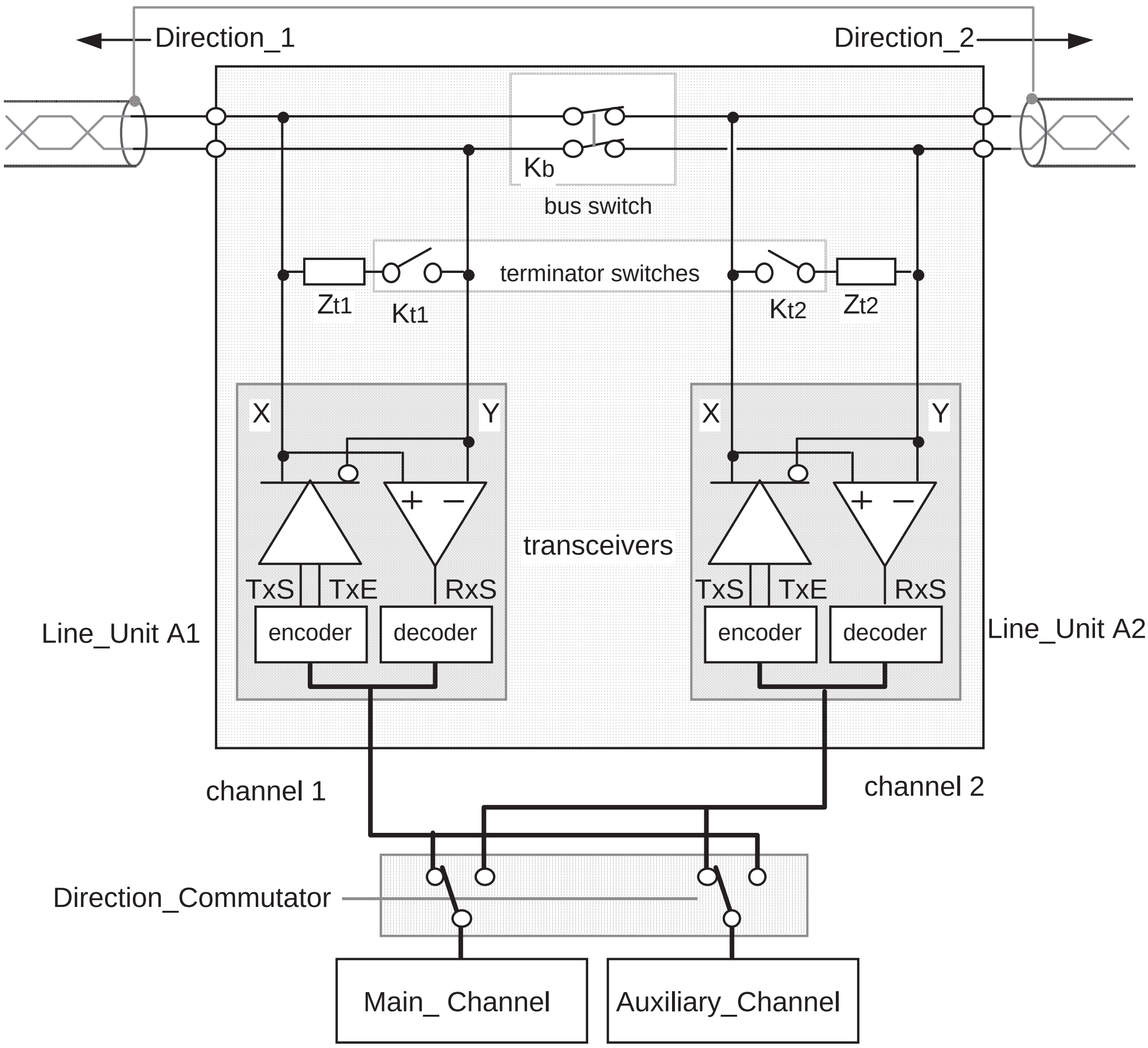


Figure 19 – Example of MAU Structure

4.4.1.1 Line Unit elements

A Line Unit shall consist of

a) a bus switch (Kb), which connects or disconnects the two directions;

b) a terminator switch for each direction (Kt1, Kt2), which inserts a terminator (Zt1, Zt2) in a node in End Setting or removes it in a node in Intermediate Setting;

c) 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;

d) 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;

e) protection circuits against over-voltage/short circuit, connected to the isolation switches (not shown in Figure 19).

NOTE 1 A node attached to a redundant medium has two Line Units.

NOTE 2 Switches Kb and Kt1 or Kt2 may be contacts of the same mechanical relay if such relays are used.

4.4.1.2 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.

4.4.1.3 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.

4.4.2 Node and switch settings

This subclause describes the node characteristics in the two settings: Intermediate Setting or End Setting.

4.4.2.1 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.

4.4.2.2 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.

4.4.3 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 20. The switchover logic allows signals to be received either from Line_A or Line_B.

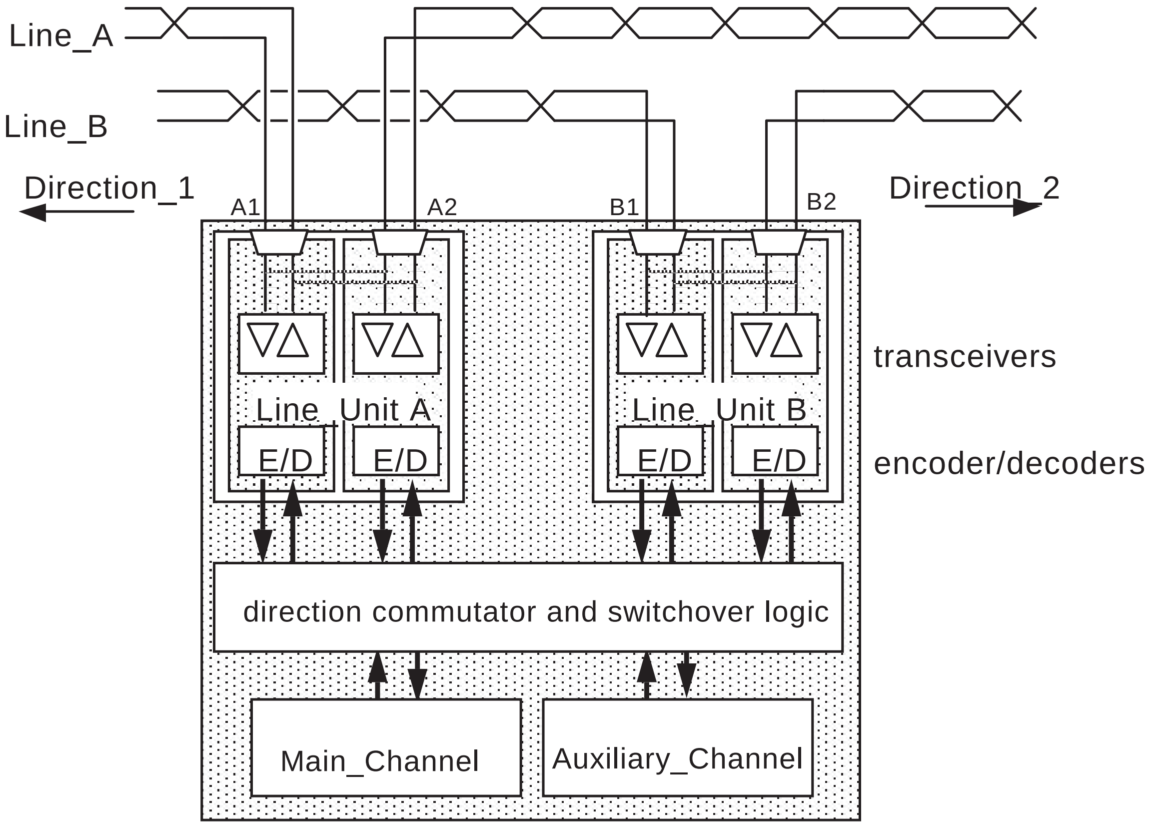


Figure 20 – Node with redundant Line Units

4.5 Line Unit specifications

Although only Line Unit A is mentioned, these specifications also apply to Line Unit B if a duplicated medium is used.

4.5.1 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 kV r.m.s. and 1,0 MΩ.

4.5.2 Insertion losses of a Line Unit

4.5.2.1 Attenuation measurement

For insertion loss measurement, the sinusoidal signal of a generator (of internal impedance = Zt) 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 Zt) at the extremity of other 20,0 m of cable attached to the points A2X and A2Y, or vice-versa, as shown in Figure 21.

The attenuation is defined as the ratio expressed in dB of two differential voltages:

a) a first voltage being set to 4,0 Vpp when the node is removed and the cable connector coupled;

b) a second voltage being measured when the node is inserted (in End Setting or Intermediate Setting, according to the test).

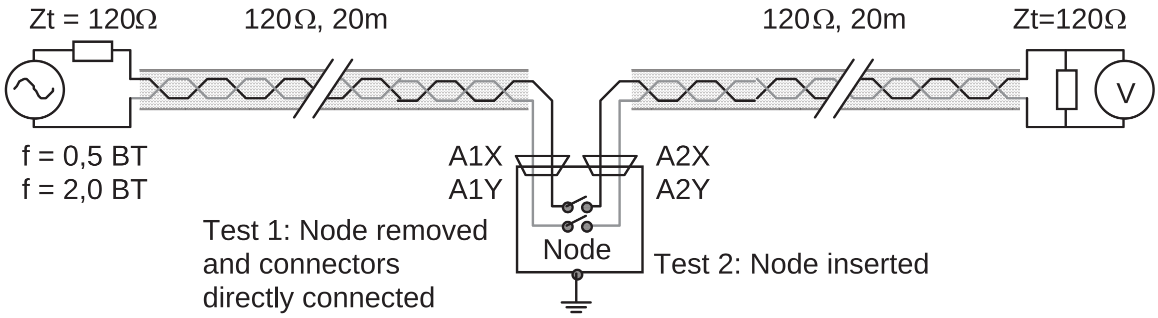


Figure 21 – Attenuation measurement

4.5.2.2 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 4.2.6.

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.

4.5.2.3 Node In Intermediate Setting

A Line Unit in Intermediate Setting (Kb closed, Kt1 and Kt2 open), either:

a) with receiver in normal operation and with transmitter in the high-impedance state, or
b) 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.

4.5.3 Switches specifications

All switches connected to the bus line (Kb, Kt, etc.):

a) shall present an isolation of at least 500,0 V r.m.s. in the open setting;
b) shall present an initial contact resistance of less than 0,050 Ω in the closed setting;
c) shall be specified for a contact resistance of less than 0,100 Ω in the closed setting after 10^7 cycles;
d) 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.

4.5.4 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 4.2.5.1 (grounded shield);
b) through the RC network specified by 4.2.5.2 (floating shield).

EXAMPLE The principle of shield attachment in a node is shown in Figure 22.

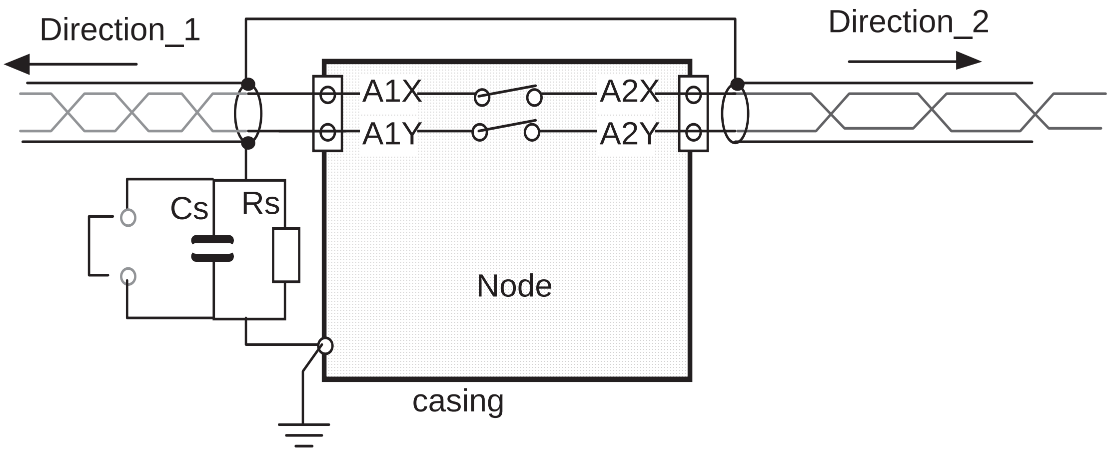


Figure 22 – Shield grounding in the Line Unit

4.5.5 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.

4.5.5.1 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 23.

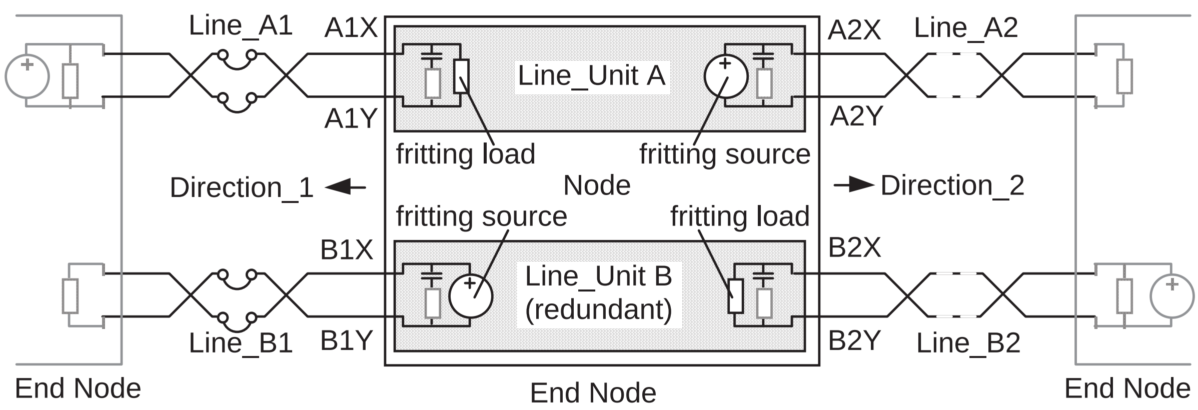


Figure 23 – Fritting source and load

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 +20 %/–10 % 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 4.2.6 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 fritting 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.

4.5.5.2 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.

4.6 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.

4.6.1 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, (Ux- Uy);
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 ±1 %, all capacitor values are ±10 %.

4.6.2 Transmitter

4.6.2.1 Transmitter load

To define the characteristics of the transmitter four test circuits are specified to simulate cables and nodes:

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 nF ± 10 % each, the resistors a value of 27,0 Ω ± 1 % each;

d) the short test circuit simulates a line failure. It consists only of a current measurement circuit;

These circuits are shown in the next figure.

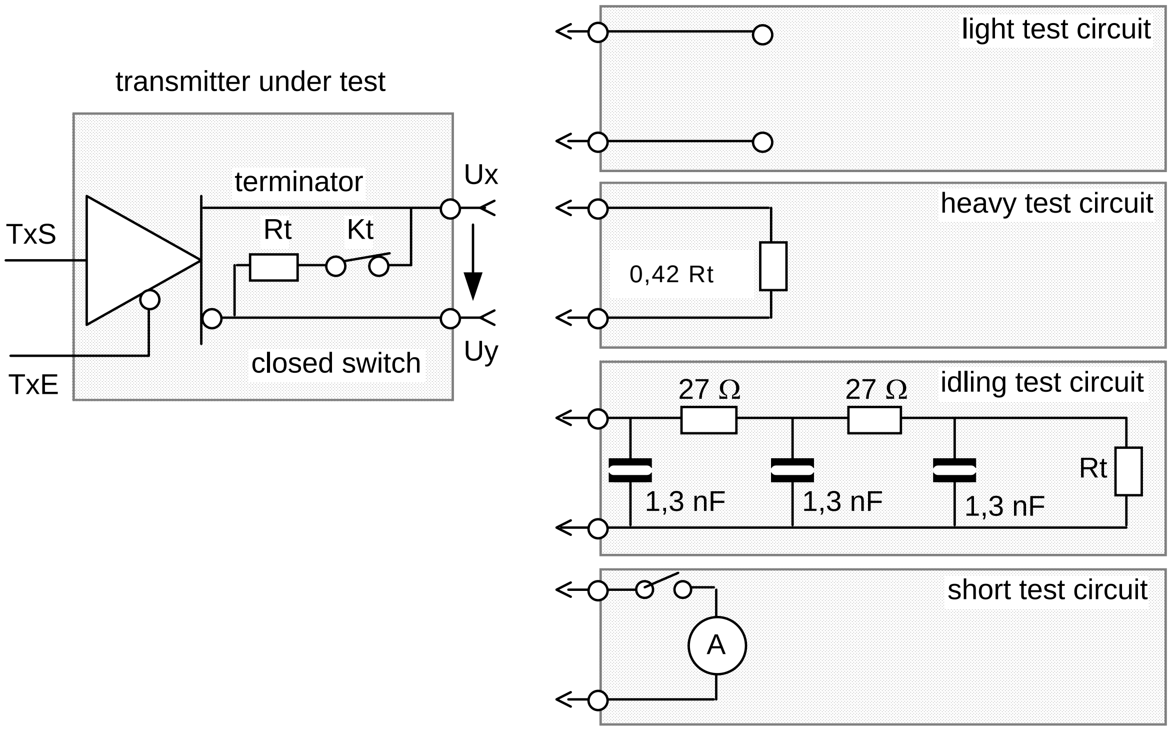


Figure 24 – Transmitter fixtures

e) the measurement is made with the node in end setting (Kb open, Kt closed);

f) the terminator of the line unit is considered in the test circuit specification.

4.6.2.2 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 (Ux-Uy) at the connection point of a node.

When connected to either the heavy and the light test circuit defined in 4.6.2.1, the transmitter shall comply with the following specifications, as shown in Figure 25:

a) the output signal shall be alternatively positive and negative;

b) the amplitude of the output signal shall be at least ±3,0 V with the heavy test circuit and at most ±7,0 V with the light test circuit;

c) 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 µs 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;

d) 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 ±2 % of one Bit Time.

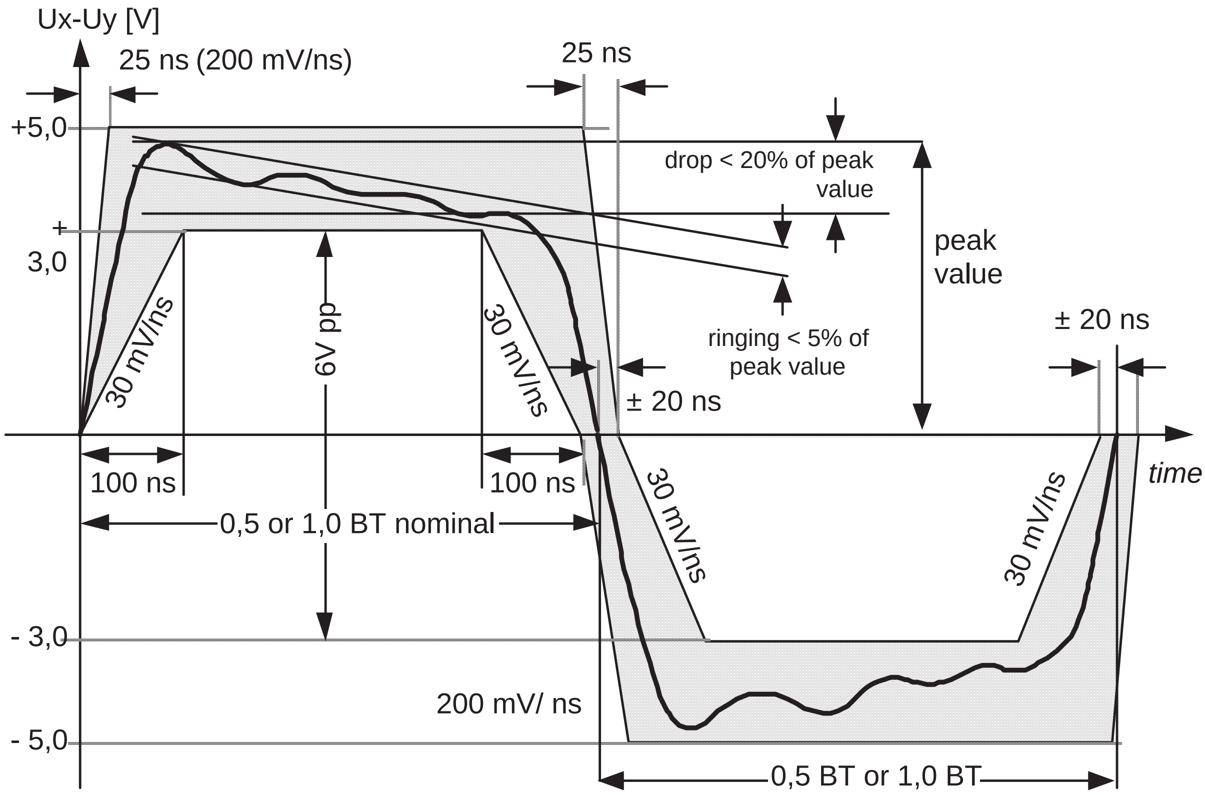


Figure 25 – Pulse wave form at transmitter

NOTE The voltage drop is expected to occur because of the fritting capacitor in series with the transformer.

4.6.2.3 Transmitter noise

Any noise generated by a transmitter which is not transmitting shall not exceed a value of 5,0 mV.r.m.s. over the frequency range 1,0 kHz to 4,0 BR.

4.6.2.4 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 4.7.1.4;

d) the transmitter drives the idling test circuit of 4.6.2.1;

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 26:

1) 100,0 ns after the last negative-to-positive transition and for 2,0 BT ± 100 ns, the output signal shall remain above 0,300 V;
2) within 3,0 BT after the last negative-to-positive transition, the output signal shall fall below 1,100 V;
3) within 20,0 µs, starting when the output signal first reaches 1,100 V, the output signal amplitude shall not exceed 0,100 V;
4) within 64,0 µs, starting when the output signal first reaches 1,100 V, the output signal amplitude shall not exceed 0,025 V.

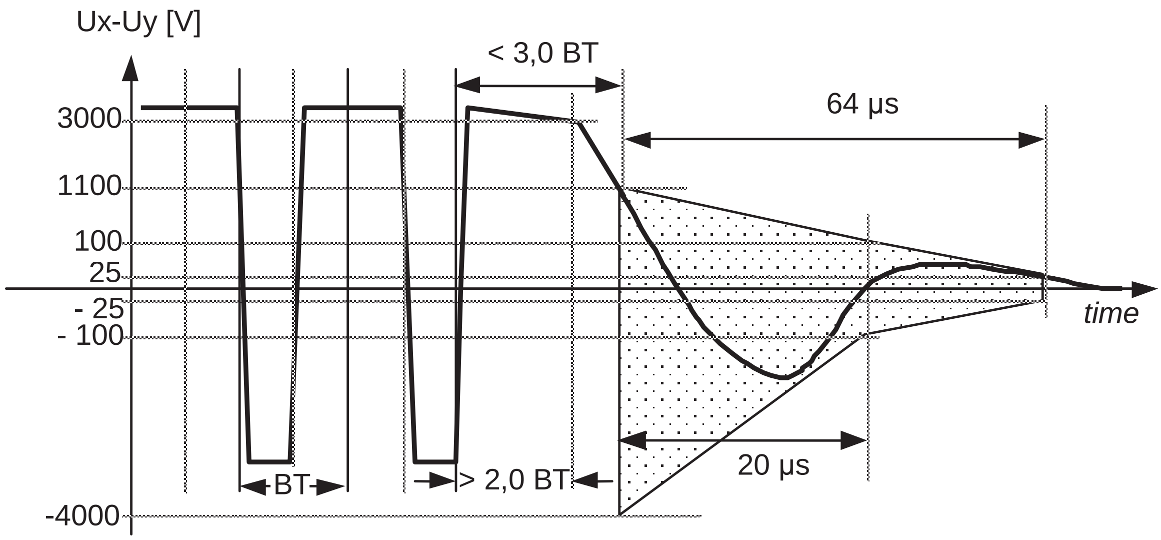


Figure 26 – Signal and idling at transmitter

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 4.7.1.4.

4.6.2.5 Transmitter fault tolerance

A transmitter, either when enabled and not, shall tolerate the application of the short test circuit (4.6.2.1) 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 1 h.

4.6.2.6 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 %.

4.6.3 Receiver specifications

4.6.3.1 Receiver signal characteristics

The receiver should decode a signal with the following shape, as shown in Figure 27, which is applied at the connection point:

a) when the amplitude of the received signal is less than 0,100 V, the slope of the received signal exceeds 2,0 mV/ns;

b) when the received 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 BT – 350,0 ns), respectively (1,0 BT – 0,350 μs), whilst its peak amplitude will vary between 0,330 V and 5,00 V;

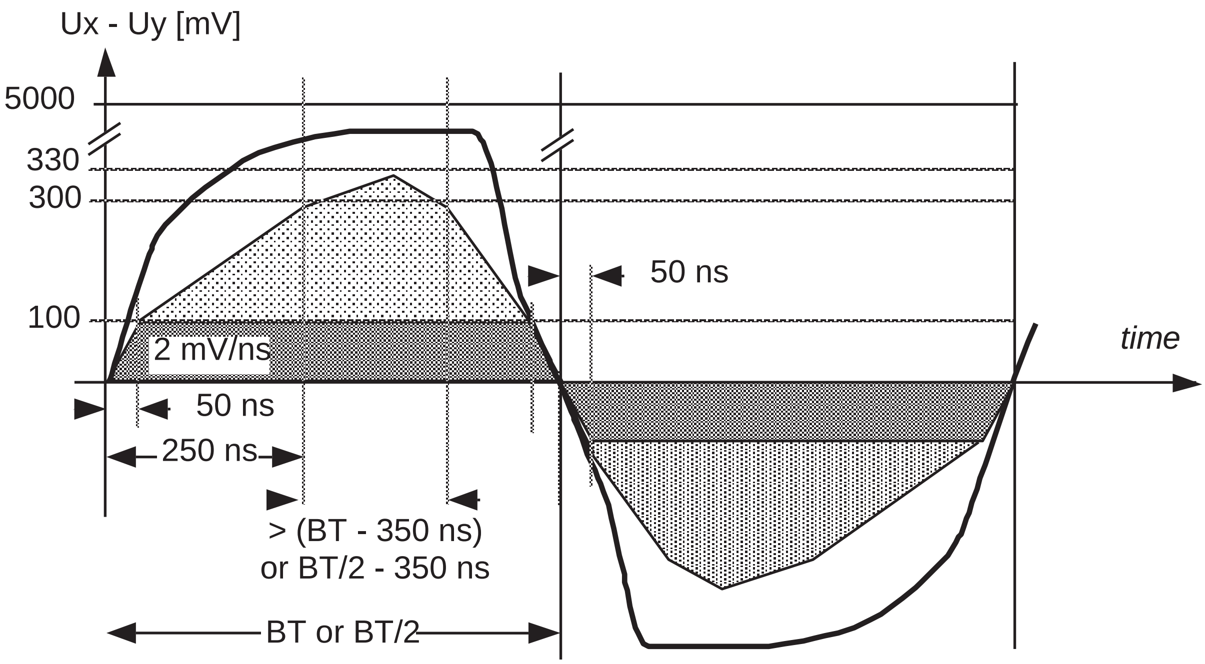


Figure 27 – Receiver signal envelope

A frame error is detected as a missing or a non-valid frame (see 4.7.1.5.3), incorrect frame size or incorrect Frame_Data bits (FCS error).

4.6.3.2 Receiver polarity

A HIGH level on TxS shall correspond to a positive differential voltage (Ux – Uy), 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 (Ux – Uy), 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.

4.6.3.3 Receiver sensitivity

A receiver receiving frames containing 64 random data bits, at the rate of 1 000 frames per second shall detect no more than three frame errors in $3 \times 3 \times 10^{+6}$ frames, when the amplitude of the received signal defined in 4.6.3.1 varies between its minimum and maximum value.

NOTE According to Figure 27, 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 4.2.3.3.

4.6.3.4 Receiver insensitivity

The receiver shall not decode a valid frame (see 4.7.1.5.3) when the received signal (4.6.3.1) is less than 0,100 V.

4.6.3.5 Receiver edge distortion

A receiver receiving frames containing 64 random data bits, at the rate of 1 000 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 ±10 % of 1,0 BT around the expected crossing, as shown in Figure 28.

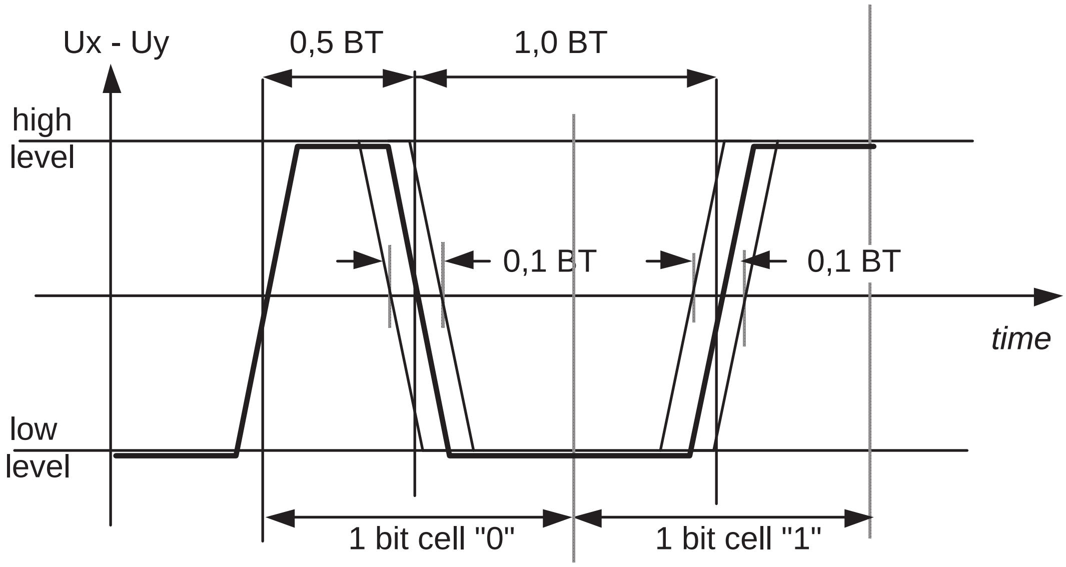


Figure 28 – Receiver edge distortion

4.6.3.6 Receiver noise rejection

A receiver receiving frames containing 64 random data bits, at the rate of 1 000 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 $3 \times 10^{+6}$ 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 V.r.m.s. 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 r.m.s.

4.7 Medium-dependent signalling

4.7.1 Frame encoding and decoding

4.7.1.1 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 29.

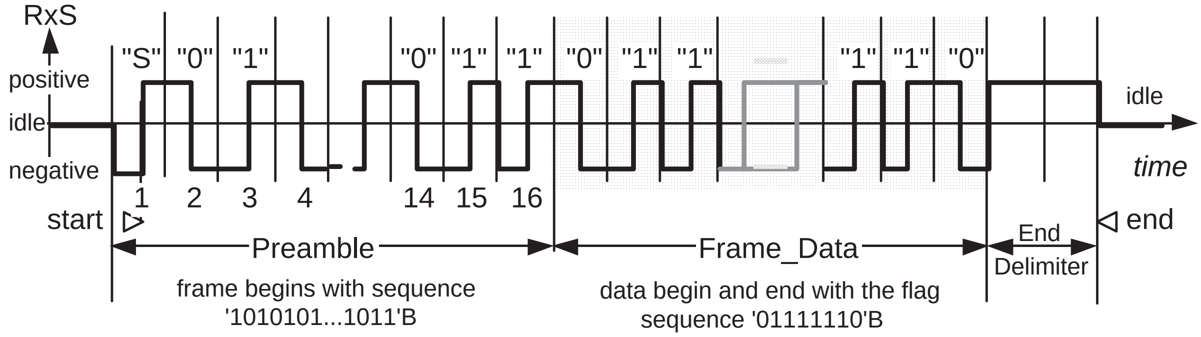


Figure 29 – Idealised frame on the line (16 bit Preamble shown)

4.7.1.2 Bit encoding

Preamble and Frame_Data bits shall be encoded by a Manchester code, as shown in Figure 30:

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.

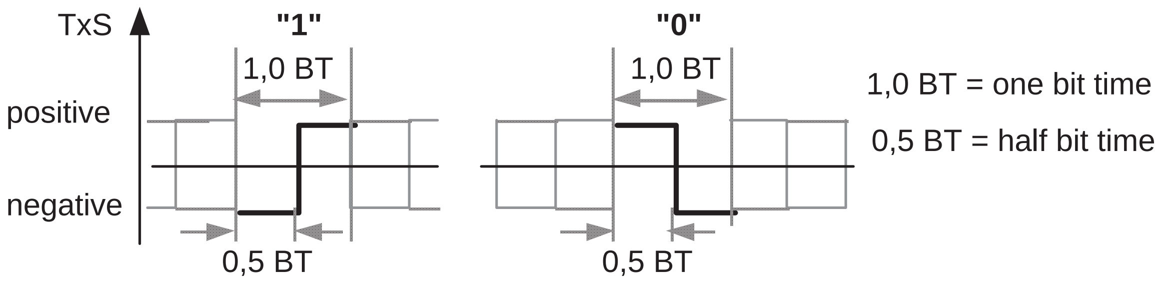


Figure 30 – Bit encoding

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.

4.7.1.3 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 31.

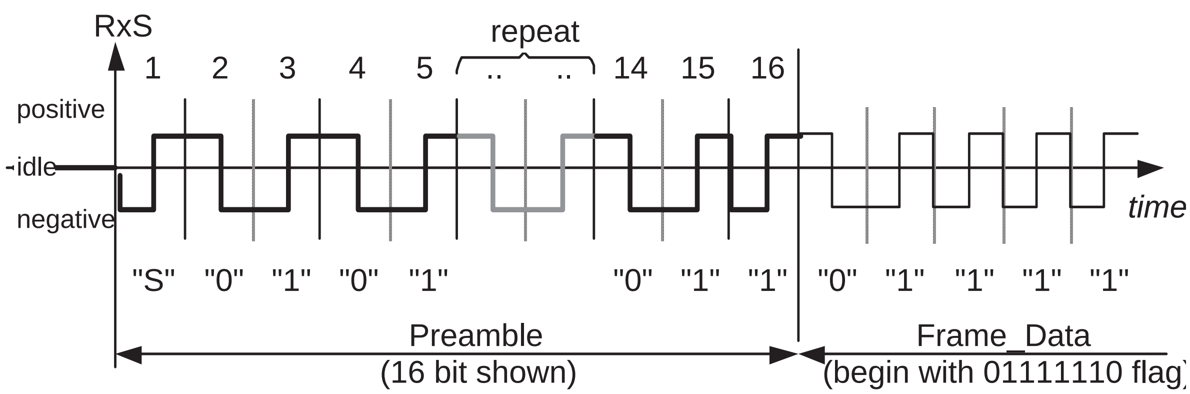


Figure 31 – Preamble

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.

4.7.1.4 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 32.

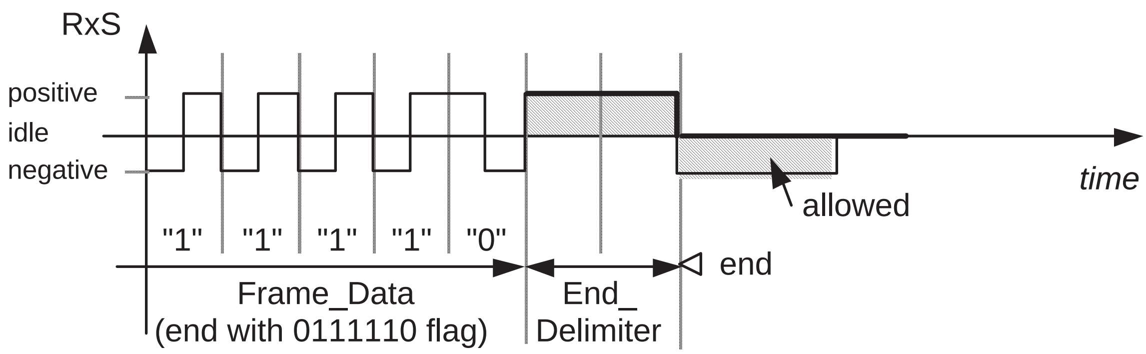


Figure 32 – End Delimiter

NOTE 1 An End Delimiter without a negative level causes an unbalance which lets the line ring (see 4.6.2.4). 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 5.2.1), the last bit of a frame is a ‘0’.

4.7.1.5 Signal quality supervision

The following specifications assume that the decoder generates two signals, called Carrier_Sense (CS) and Signal_Quality_Error (SQE), for signal quality supervision and redundancy switchover.

4.7.1.5.1 Carrier_Sense

The decoder shall assert CS within 0,5 BT after it detects the last received bit of a Preamble according to 4.7.1.3.

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.

4.7.1.5.2 Signal_Quality_Error

The decoder shall negate SQE within 0,5 BT after it detects the last received bit of a Preamble according to 4.7.1.3.

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.

4.7.1.5.3 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 33.

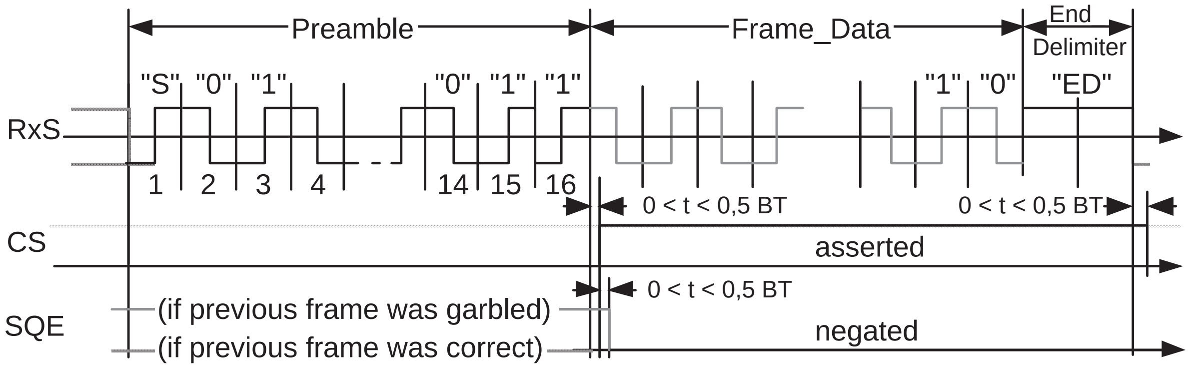


Figure 33 – Valid frame, RxS, CS and SQE signals

For the purpose of redundancy control, a valid frame shall consist of a Preamble followed by a sequence of at least eight data bits.

4.7.1.5.4 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 34.

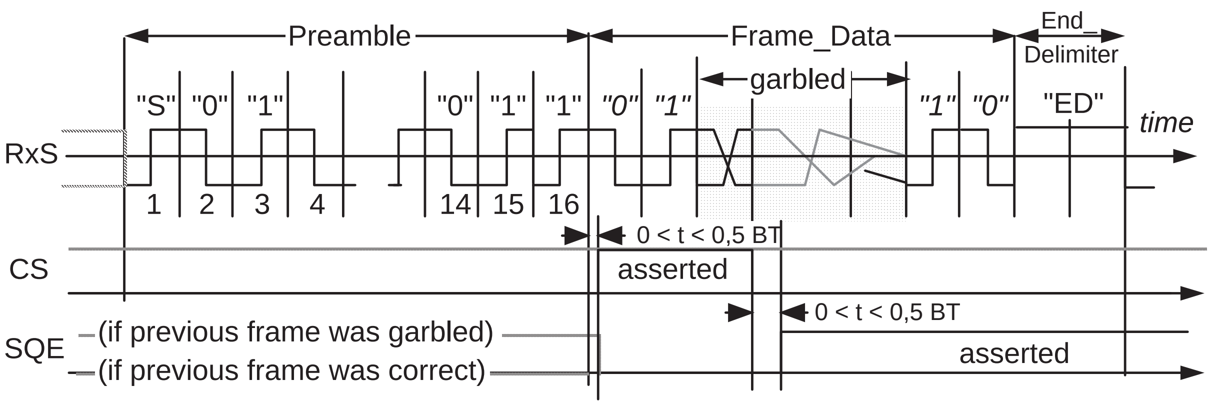


Figure 34 – Garbled frame, RxS, CS, SQE signals

If SQE becomes active, the decoder shall ignore all data until the next Preamble is received.

4.7.2 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.

4.7.2.1 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.

4.7.2.2 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 35.

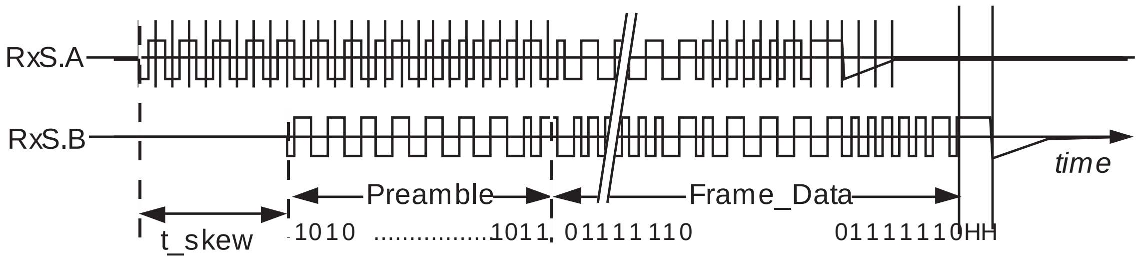


Figure 35 – Redundant Lines (as seen at a receiver)

4.7.2.3 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 µs.

4.7.2.4 Redundant reception

4.7.2.4.1 Skew at reception

A receiver shall permit a maximum skew at the receiver, T_skew_r, of 32,0 µs.

4.7.2.4.2 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 4.7.1.5.3.

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 36.

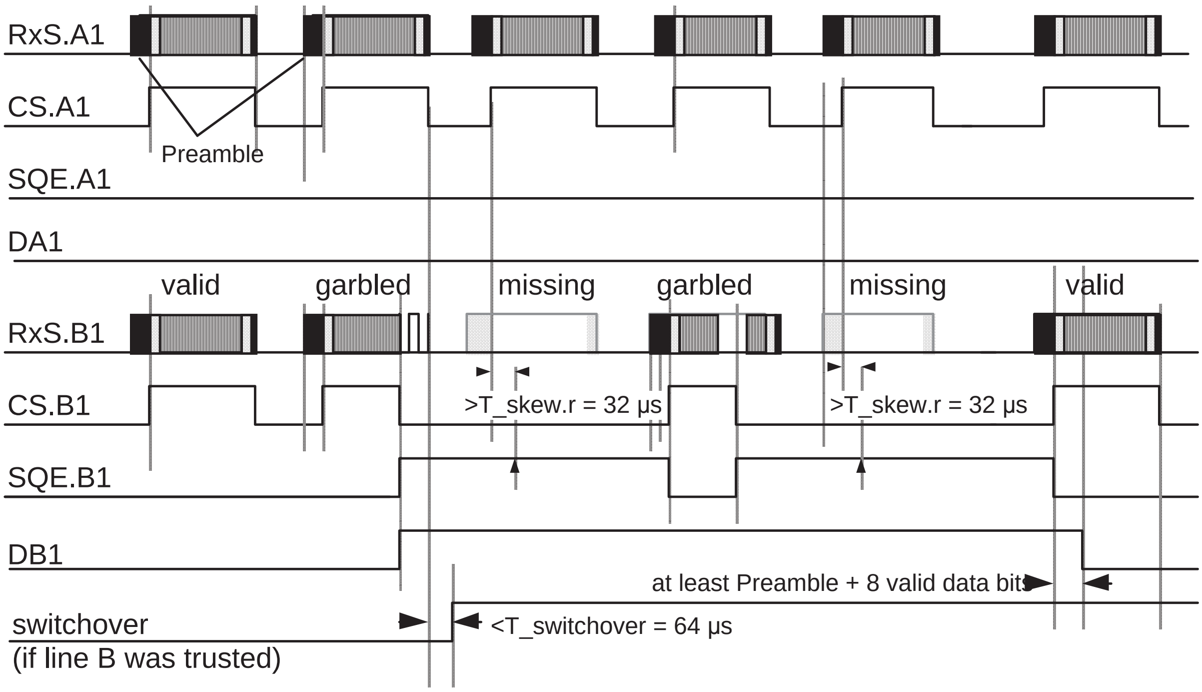


Figure 36 – Line_Disturbance signals

4.7.2.5 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_switchover = 64,0 µs:

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).

4.7.2.6 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 5.5.2.2).

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.

4.7.3 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 ITU-T Recommendation V.24, for each transceiver separately, and of additional control signals, as specified in Table 7.

Table 7 – Signals of the Line Unit Interface

Name	Designation	Clause of ITU-T Rec. V.24	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 4.7.1.5.2
CS	Carrier_Sense	Not V.24	from Line Unit	as specified in 4.7.1.5.1
Kx	Switch control signals	Not V24	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.

5 Link Layer Control

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.

5.2 Frames and telegrams

5.2.1 Frame_Data format

The Frame_Data format shall conform to the HDLC format defined by ISO/IEC 13239, as shown in Figure 37.

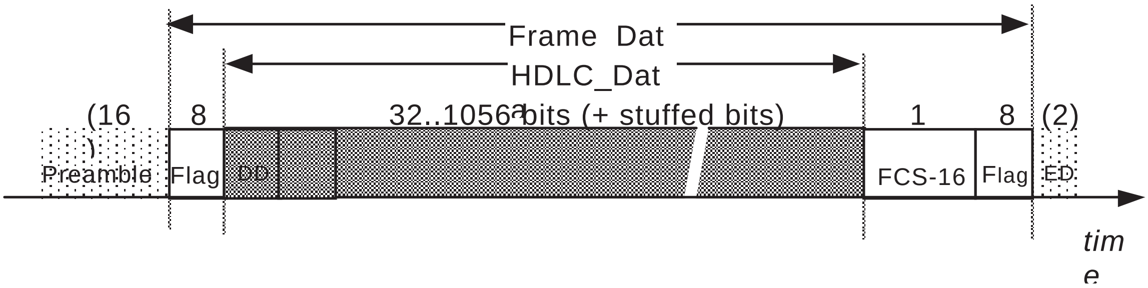


Figure 37 – HDLC Frame structure

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 13239.

The opening flag shall be followed by the HDLC Data, consisting of at least 32 bits and at most of 1 056 bits (not counting the stuffed bits foreseen by ISO/IEC 13239).

The HDLC Data shall have (as a restriction to ISO/IEC 13239) an integer number of octets.

The first octet of the HDLC Data shall be the 8-bit Destination Device specified in 5.1.

The second octet of the HDLC Data shall not be interpreted as the control field of ISO/IEC 13239.

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 13239 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 13239.

A frame is considered as a sequence of octets. The least significant bit of each octet shall be transmitted first, as ISO/IEC 13239 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 the most significant bit first due to an exception in ISO/IEC 13239.

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 %.

5.2.2 Telegram timing

5.2.2.1 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 38.

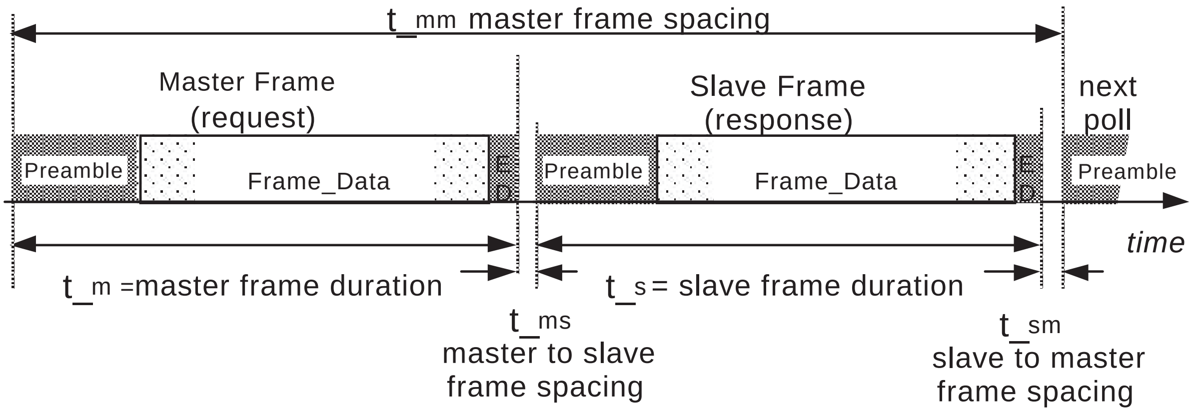


Figure 38 – Telegram timing

5.2.2.2 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 39.

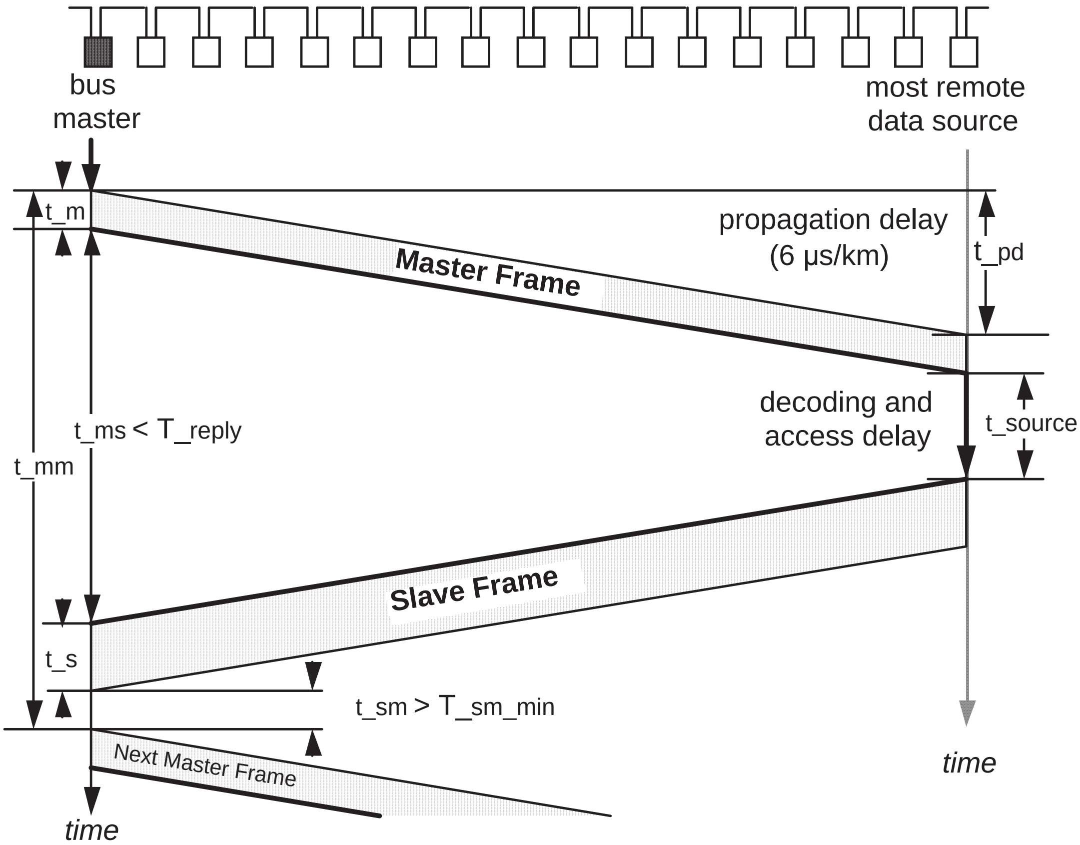


Figure 39 – Example of Interframe spacing

The worst-case reply relay for a given application, T_reply , is evaluated as:

T_reply [s] = 2 × T_pd + T_source_max

T_source_max accounts for the decoding of the Master Frame and reply at the source (see 5.2.2.4.2);

T_pd is the worst case propagation delay of a frame between the End Nodes in a given application (see 4.2.3).

The time-outs of a frame are calculated as follows:

Bit times	Main channel	Auxiliary channel
application data bits	1 024 bits	16 bits
header	32 bits	32 bits
CRC	16 bits	16 bits
total bits	1 072 bits	64 bits
bit stuffing (worst case × 1,2)	1 286 bits	77 bits
2 × flag	16 bits	16 bits
end delimiter	2 bits	2 bits
preamble maximum size	32 bits	32 bits
Total bits	1 336 bits	127 bits
duration at 1,0 Mbit/s	1 336,0 μs	127,0 μs
2 × propagation delay	120,0 μs	120,0 μs
response time of slave	300,0 μs	800,0 μs
Total	1 756,0 μs	1 047,0 μs

5.2.2.3 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.

5.2.2.4 Transmitted frame spacing

5.2.2.4.1 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_main_max = 1,756 ms since the end of its transmitted Master Frame, and it may start sending its next Master Frame T_sm_min = 0,064 ms after the end of the received Slave Frame or after the time-out (1,756 ms + 0,064 ms = 1,820 ms) elapsed, as Figure 40 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_aux_max = 1,047 ms, and it may start sending its next Master Frame T_ms_min = 0,064 ms after the end of the received Detect Response or after the time-out (1,047 ms + 0,064 ms = 1,111 ms) elapsed.

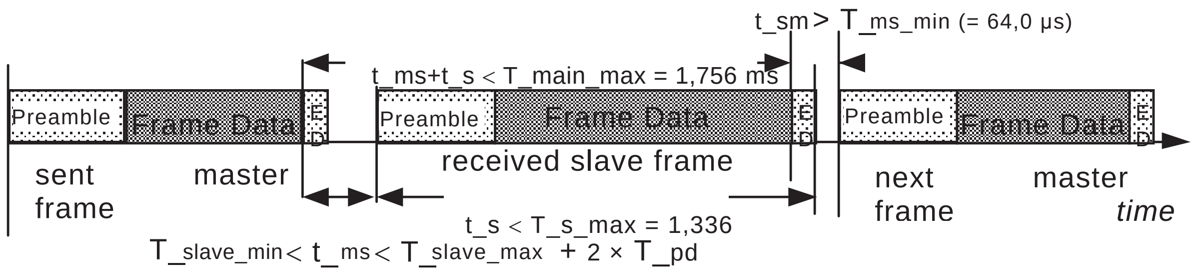


Figure 40 – Frame spacing measured at the master side

5.2.2.4.2 Slave side

The addressed slave shall start transmitting its frame, as shown in Figure 41:

- not earlier than T_source_min = 64,0 µs after receiving the end of the Master Frame; and
- not later than T_source_max = 0,300 ms after receiving the end of the Master Frame, except for a Detect Response for which this time is raised to 0,600 ms during inauguration and 0,800 ms during regular operation.

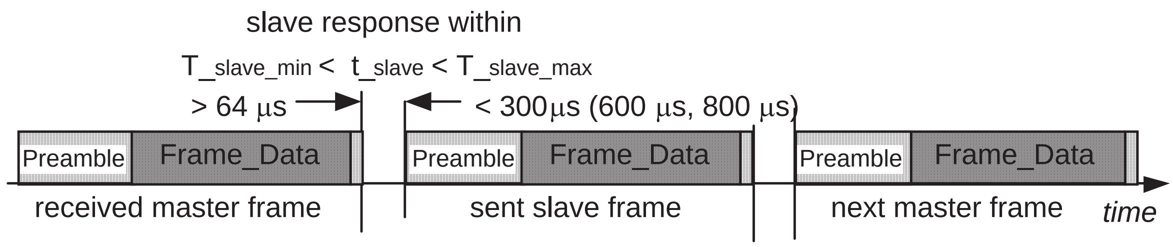


Figure 41 – Frame spacing at the slave

5.2.3 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 42:

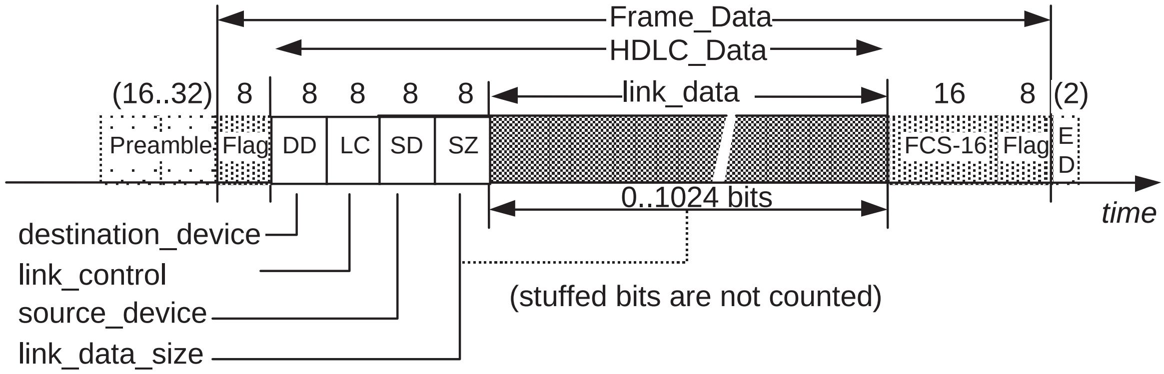
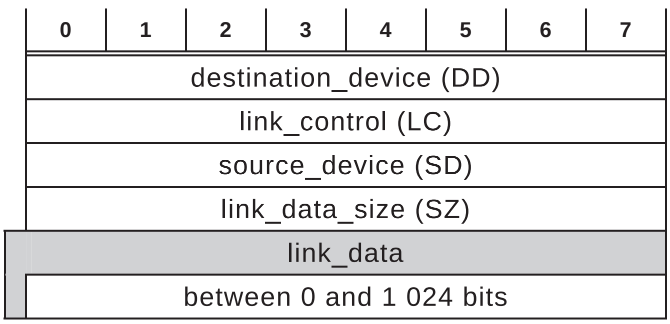


Figure 42 – HDLC Data format

The HDLC Data shall consist of the following fields, shown in Figure 43:

```
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 1 024 bits of
                                              data; see 5.3.1 for details.
}
```



NOTE The aliases DD, LC, SD, SZ are used in the figures to save space.

Figure 43 – Format of HDLC Data

5.2.4 Link Control Field

The Link Control field distinguishes:

a) Requests (Master Frames) from
b) Responses (Slave Frames).

The Link Control Field distinguishes three types of telegrams:

c) Process Data telegrams used to update the distributed Process data base;
d) Message Data telegrams used for message transfer;
e) 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:

f) ‘A_bit’: (Attention) Message Data transmission required;
g) ‘C_bit’: (Change) node status change;
h) ‘I_bit’: (Inhibit) inauguration not permitted
i) ‘RI_bit’: (Remote Inhibit) remote inauguration not permitted.

Link Control shall be encoded into 8 bits as specified in Table 8.

Table 8 – Link Control encoding

		Encoding							
	Frame Type	7	6	5	4	3	2	1	0
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 8 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_inh             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
        }
      }
    }
  }
```

5.2.5 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.

5.2.6 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).

5.3 Telegram formats and protocols

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:

```
HDLC_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.

5.3.2 Process Data

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 44.

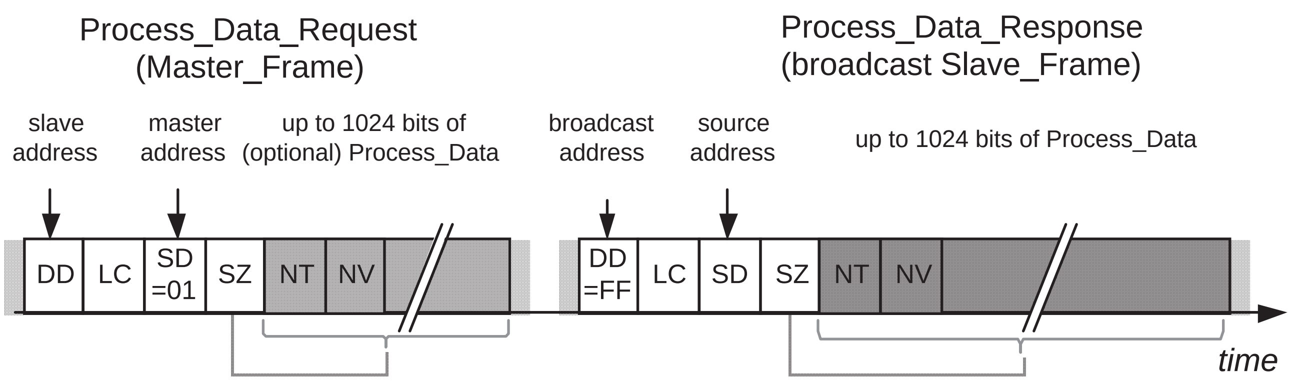


Figure 44 – Process Data telegram

5.3.2.2 Process Data Request

A Process Data Request shall have the following format, as shown in Figure 45:

```
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
```

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">destination_device</td></tr>
<tr><td colspan="8">link_control</td></tr>
<tr><td colspan="8">source_device</td></tr>
<tr><td colspan="8">link_data_size</td></tr>
<tr><td colspan="8">ARRAY [link_data_size] OF</td></tr>
<tr><td colspan="8">WORD8</td></tr>
</table>

Figure 45 – Format of Process Data Request

5.3.2.3 Process Data Response

A Process Data Response shall have the following format, as shown in Figure 46:

```
Process_Data_Response::= RECORD
  {
  destination_device    UNSIGNED8            -- destination_device =
                                                broadcast
  link_control          Link_Control         -- Process_Data_Response
  source_device         UNSIGNED8            -- 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.
  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							
ARRAY [link_data_size] OF							
WORD8							

Figure 46 – Format of Process Data Response

5.3.3 Message Data

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 47.

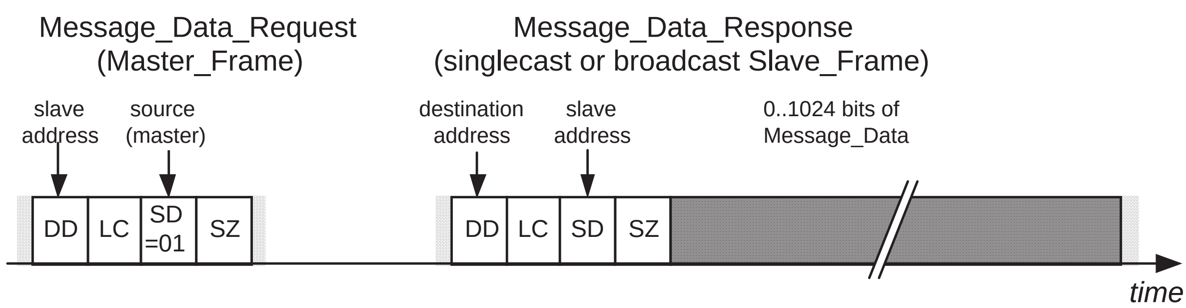


Figure 47 – Message Data telegram

NOTE The Message Data structure is specified in Clause 6.

5.3.3.2 Message Data Request

A Message Data Request shall have the following format, as shown by Figure 48:

```
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
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							

Figure 48 – Format of Message Data Request

5.3.3.3 Message Data Response

A Message Data Response shall have the following format, as shown by Figure 49:

```
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 6
  }
```

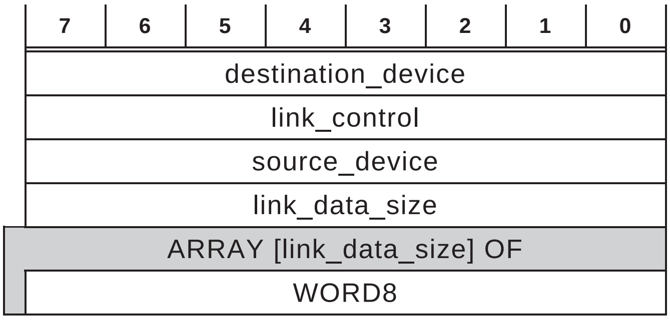


Figure 49 – Format of Message Data Response

5.3.4 Supervisory Data

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 50.

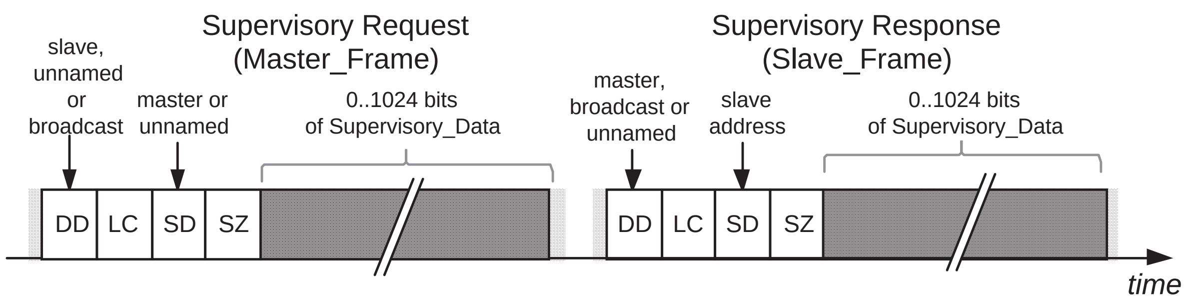


Figure 50 – Supervisory telegram

Over the Auxiliary Channel, an End Node may play the role of a master.

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
  }
```

5.3.5 Detection telegram

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 51.

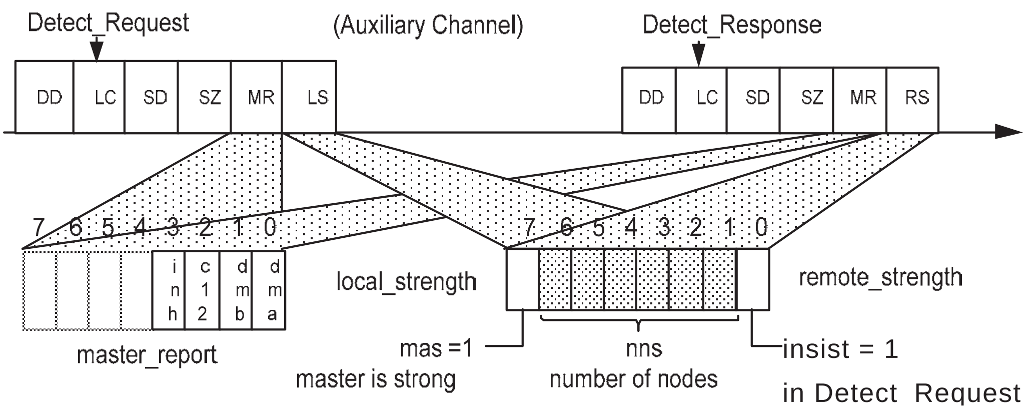


Figure 51 – Detection telegram

5.3.5.2 Detect_Request

Detect_Request shall have the following format, shown in Figure 52:

```
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  5.5.2.5
  local_strength        Composition_Strength   -- copy of LocStr of
                                               requesting node( 5.5.2.4)
                                              -- 'ins' is set to '1'.
  }
```

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">destination_device</td></tr>
<tr><td colspan="8">link_control</td></tr>
<tr><td colspan="8">source_device</td></tr>
<tr><td colspan="8">link_data_size</td></tr>
<tr><td colspan="8">master_report</td></tr>
<tr><td>mas</td><td colspan="6">nns</td><td>ins</td></tr>
</table>

Figure 52 – Format of Detect Request

5.3.5.3 Detect Response

Detect Response shall have the following format, as shown in Figure 53:

```
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 (
                                               5.5.2.4)
  }
```

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">destination_device</td></tr>
<tr><td colspan="8">link_control</td></tr>
<tr><td colspan="8">source_device</td></tr>
<tr><td colspan="8">link_data_size</td></tr>
<tr><td colspan="8">master_report</td></tr>
<tr><td>mas</td><td colspan="6">nns</td><td>ins</td></tr>
</table>

Figure 53 – Format of Detect Response

5.3.6 Presence telegram

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 54.

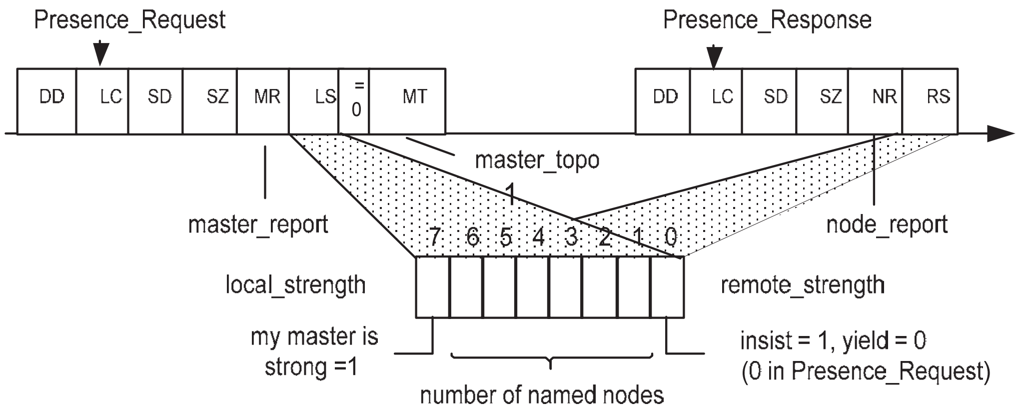


Figure 54 – Presence telegram

5.3.6.2 Presence Request

Presence Request shall have the following format, shown in Figure 55:

```
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 5.5.2.5
  local_strength        Composition_Strength    -- copy of LocStr of master,
                                                'ins' is '0'.
  reserved1             WORD4  (=0)           -- reserved, =0
  master_topo           Master_Topo           -- see 5.5.2.7
  }
```

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">destination_device</td></tr>
<tr><td colspan="8">link_control</td></tr>
<tr><td colspan="8">source_device</td></tr>
<tr><td colspan="8">link_data_size</td></tr>
<tr><td colspan="8">master_report</td></tr>
<tr><td>mas</td><td colspan="6">nns</td><td>ins</td></tr>
<tr><td colspan="4">reserved1</td><td colspan="4" rowspan="2">master_topo</td></tr>
</table>

Figure 55 – Format of Presence Request

5.3.6.3 Presence Response

Presence Response shall have the following format, as shown in Figure 56:

```
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 5.5.2.2
  remote_strength       Composition_Strength   -- copy of RemStr of End
                                               Node
                                               'ins' = '1' indicates that
                                               the other composition
                                               insists.
  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							
node_report							
mas	nns						ins

Figure 56 – Format of Presence Response

5.3.7 Status telegram

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 57.

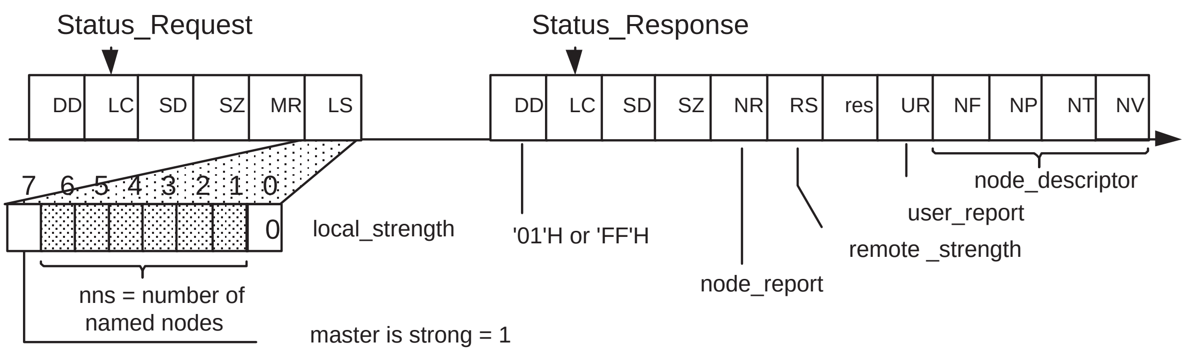


Figure 57 – Status telegram

5.3.7.2 Status Request

Status Request shall have the following format, shown in Figure 58:

```
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 5.5.2.5
  local_strength        Composition_Strength   -- LocStr of master, ‘ins’
                                                  is 0
  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							
master_report							
mas	nns						ins

Figure 58 – Format of Status Request

5.3.7.3 Status Response

Status Response shall have the following format, shown in Figure 59:

```
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 5.5.2.2
  remote_strength       Composition_Strength   -- remote composition
                                                  strength in end node,
                                                  0 in intermediate node.
  reserved1             WORD8 (=0)             -- reserved, =0
  user_report           User_Report            -- see 5.5.2.3
  node_descriptor       Node_Descriptor        -- see 5.5.2.1
  }
```

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">destination_device</td></tr>
<tr><td colspan="8">link_control</td></tr>
<tr><td colspan="8">source_device</td></tr>
<tr><td colspan="8">link_data_size</td></tr>
<tr><td colspan="8">node_report</td></tr>
<tr><td>mas</td><td colspan="6">nns</td><td>ins</td></tr>
<tr><td colspan="8">reserved1</td></tr>
<tr><td colspan="8">user_report</td></tr>
<tr><td colspan="8">node_frame_size</td></tr>
<tr><td colspan="5">reserved1</td><td colspan="3">node_period</td></tr>
<tr><td colspan="8">node_type</td></tr>
<tr><td colspan="8">node_version</td></tr>
</table>

Figure 59 – Format of Status Response

5.3.8 Set to Intermediate telegram

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 60.

Figure 60 – Set-to-Intermediate telegram

5.3.8.2 SetInt Request

SetInt Request shall have the following format, shown in Figure 61:

```
SetInt_Request::= RECORD
  {
  destination_device    UNSIGNED8          -- node address
  link_control          Link_Control       -- SetInt_Request
  source_device         UNSIGNED8          -- 'master' address
  link_data_size        UNSIGNED8          -- =0
  }
```

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">destination_device</td></tr>
<tr><td colspan="8">link_control</td></tr>
<tr><td colspan="8">source_device</td></tr>
<tr><td colspan="8">link_data_size</td></tr>
</table>

Figure 61 – Format of SetInt Request

5.3.8.3 SetInt Response

SetInt Response shall have the following format, shown in Figure 62:

```
SetInt_Response::= RECORD
  {
  destination_device    UNSIGNED8           -- 'master' address
  link_control          Link_Control        -- SetInt_Response
  source_device         UNSIGNED8           -- node address
  link_data_size        UNSIGNED8           -- =0
  }
```

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">destination_device</td></tr>
<tr><td colspan="8">link_control</td></tr>
<tr><td colspan="8">source_device</td></tr>
<tr><td colspan="8">link_data_size</td></tr>
</table>

Figure 62 – Format of SetInt Response

5.3.9 Naming telegram

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 63.

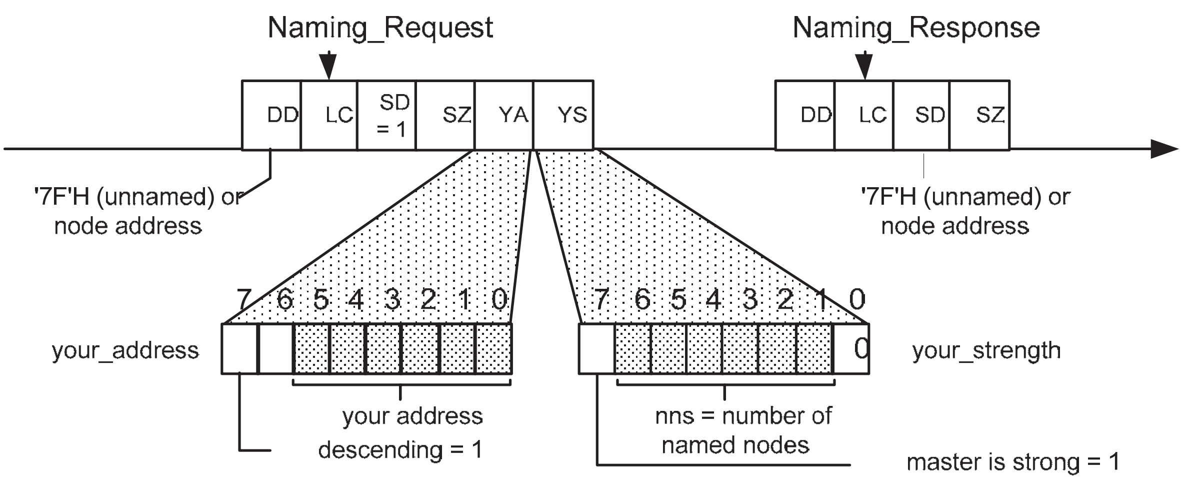


Figure 63 – Naming telegram

5.3.9.2 Naming Request

Naming Request shall have the following format, shown in Figure 64:

```
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
  }
```

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">destination_device</td></tr>
<tr><td colspan="8">link_control</td></tr>
<tr><td colspan="8">source_device</td></tr>
<tr><td colspan="8">link_data_size</td></tr>
<tr><td>dir1</td><td>rsv1</td><td colspan="6">your_address</td></tr>
<tr><td>mas</td><td colspan="6">nns</td><td>ins</td></tr>
</table>

Figure 64 – Format of Naming Request

5.3.9.3 Naming Response

Naming Response shall have the following format, shown in Figure 65:

```
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 (5.3.9)
  link_data_size        UNSIGNED8            -- =0
  }
```

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">destination_device</td></tr>
<tr><td colspan="8">link_control</td></tr>
<tr><td colspan="8">source_device</td></tr>
<tr><td colspan="8">link_data_size</td></tr>
</table>

Figure 65 – Format of Naming Response

5.3.10 Unname telegram

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 66 (there is no Unname Response).

Figure 66 – Unnaming telegram

5.3.10.2 Unname Request

Unname Request shall have the following format, shown in Figure 67:

```
Unname_Request::= RECORD
  {
  destination_device    UNSIGNED8           -- 'broadcast' address
  link_control          Link_Control        -- Unname_Request
  source_device         UNSIGNED8           -- 'master' address
  link_data_size        UNSIGNED8           -- =0
  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							

Figure 67 – Format of Unname Request

5.3.11 Set to End telegram

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 68.

Figure 68 – Set to End telegram

5.3.11.2 SetEnd Request

SetEnd Request shall have the following format, shown in Figure 69:

```
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
                                               (5.5.2.4)
  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							
reserved1							
mas	nns						ins

Figure 69 – Format of SetEnd Request

5.3.11.3 SetEnd Response

SetEnd Response shall have the following format, shown in Figure 70:

```
SetEnd_Response::= RECORD
  {
  destination_device   UNSIGNED8              -- ‘master’ address
  link_control         Link_Control           -- SetEnd_Response
  source_device        UNSIGNED8              -- node address
  link_data_size       UNSIGNED8              -- = 0
  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							

Figure 70 – Format of SetEnd Response

5.3.12 Topography telegram

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 71:

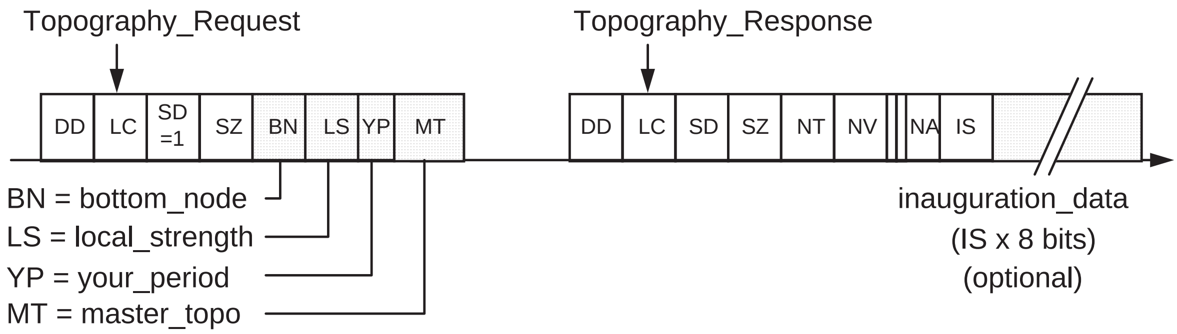


Figure 71 – Topography telegram

5.3.12.2 Topography Request

Topography Request shall have the following format, shown in Figure 72:

```
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
                                                 (5.5.2.1 and 5.4.2)
  master_topo           Master_Topo           -- see 5.5.2.7
  }
```

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">destination_device</td></tr>
<tr><td colspan="8">link_control</td></tr>
<tr><td colspan="8">source_device</td></tr>
<tr><td colspan="8">link_data_size</td></tr>
<tr><td colspan="8">bottom_node</td></tr>
<tr><td>mas</td><td colspan="6">nns</td><td>ins</td></tr>
<tr><td colspan="4">your_period</td><td colspan="4"></td></tr>
<tr><td colspan="8">master_topo</td></tr>
</table>

Figure 72 – Format of Topography Request

5.3.12.3 Topography Response

Topography Response shall have the following format, shown in Figure 73:

```
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
  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							
node_type							
node_version							
sam	rsv1	node_address					
inaug_data_size							
ARRAY [inaug_data_size] OF							
WORD8							

Figure 73 – Format of Topography Response

NOTE The redundancy of ‘inaug_data_size’ with ‘link_data_size’ is intentional and used for plausibility checks.

5.4 Medium allocation

5.4.1 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 5.5. Master selection is not part of the medium allocation described here.

5.4.1.1 Basic Period

5.4.1.1.1 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 74:

a) a Periodic Phase used to transmit periodic data and

b) a Sporadic Phase used for transmitting:

- Supervisory Data; and/or
- Message Data.

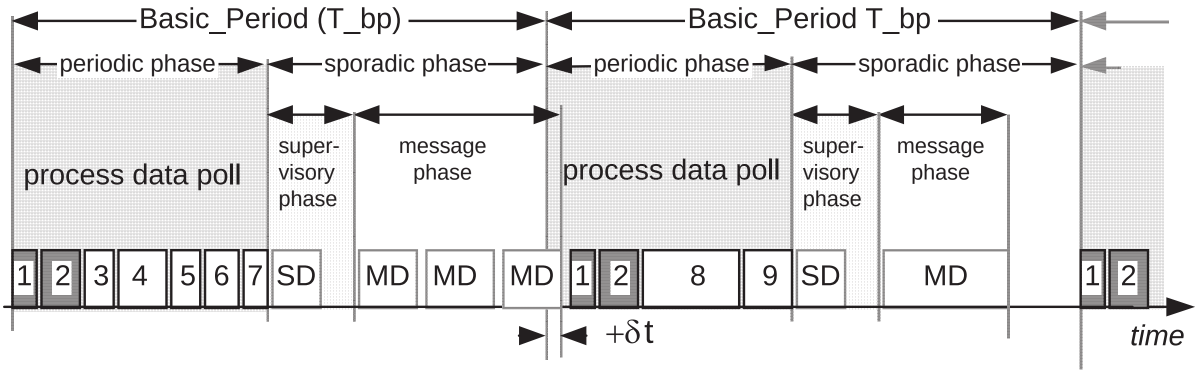


Figure 74 – Structure of the Basic Period

5.4.1.1.2 Duration of the Basic Period

The duration of the Basic Period shall be T_bp = 25,0 ms ± 1,0 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.

5.4.2 Periodic Phase

5.4.2.1 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, so that T_ip = (2n × 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 5.5.2.1) during Inauguration.

The longest Individual Period of any node on the bus defines the Macro Period.

5.4.2.2 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.

5.4.2.3 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.

5.4.2.4 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 5.5.4.9.3.

5.4.3 Sporadic phase

5.4.3.1 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.

5.4.3.2 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.

5.4.3.3 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).

5.4.3.4 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.

5.5 Inauguration

5.5.1 General

5.5.1.1 Address Allocation

The inauguration procedure shall allocate each node an address, as shown in Figure 75:

- nodes in Direction_1 from the master are sequentially numbered in descending order, starting with 63, the last named node being the bottom node;
- nodes in Direction_2 from the master are sequentially numbered in ascending order starting with 02, the last named node being the top node.

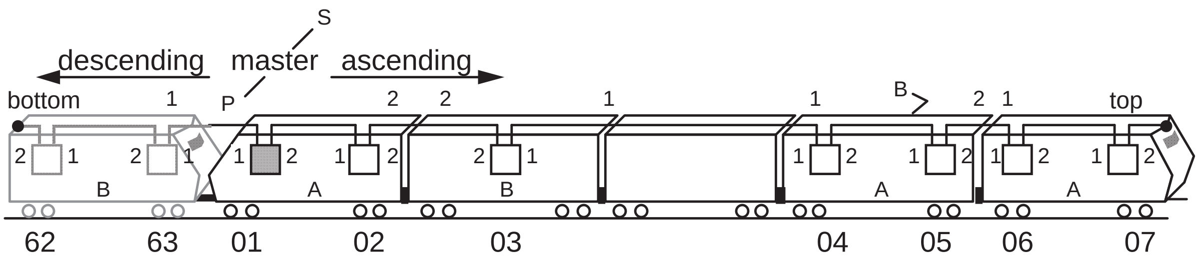


Figure 75 – Node position numbering

5.5.1.2 Node ranking

The application may implement different mastership strategies by ranking a node as either:

strong node, or

weak node or

slave node.

5.5.1.3 Strong node

A strong node is promoted by the application to become the master.

A strong node exercising mastership is called a strong master.

A strong node may be demoted to weak node by the application. If it was strong master, it will signal its demoting to all nodes and it will remain in control of the bus as weak master until a strong node is appointed.

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.

NOTE 2 A strong node allows to tie mastership to certain application functions, for instance with the leading vehicle. Such a tying is required for the inclusion of Process Data in the Process_Data_Request frames (5.3.1).

5.5.1.3.1 Weak node

A weak node is a node which the application allows to become master.

A weak node exercising mastership is called a weak master.

Normally, the application appoints several or all nodes as weak nodes in a composition.

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.

5.5.1.3.2 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.

5.5.2 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 5.6.4.

5.5.2.1 Node Descriptor

Each node shall implement a Node Descriptor to identify its characteristics.

The Node Descriptor shall be represented by the following data structure, as shown in Figure 76:

```
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.
  }
```

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">node_frame_size</td></tr>
<tr><td colspan="5">reserved1</td><td colspan="3">node_period</td></tr>
<tr><td colspan="8">node_type</td></tr>
<tr><td colspan="8">node_version</td></tr>
</table>

Figure 76 – 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 UIC CODE 556.

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 (5.6.2.2).

5.5.2.2 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 77.

```
Node_Report::= BITSET8
  {
  da1          (0)     -- Line_A1 disturbed, copies DA1
                          of 4.7.2.4
  da2          (1)     -- Line_A2 disturbed, copies DA2
                          of 4.7.2.4
  dB1          (2)     -- Line_B1 disturbed, copies dB1
                          of 4.7.2.4
  dB2          (3)     -- Line_B2 disturbed, copies dB2
                          of 4.7.2.4
  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.

 }
```

7	6	5	4	3	2	1	0
da1	da2	db1	db2	int	dsc	slp	sam

Figure 77 – Format of Node Report

5.5.2.3 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 78:

```
User_Report::= WORD8                    -- application-defined
```

7	6	5	4	3	2	1	0
ur7	ur6	ur5	ur4	ur3	ur2	ur1	ur0

Figure 78 – Format of User Report

NOTE An application may set individual bits of User Report through link layer services.

5.5.2.4 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 79.

```
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
  }
```

7	6	5	4	3	2	1	0
mas	nns						ins

Figure 79 – Format of Composition Strength

To simplify strength comparisons, the local or remote composition strength shall be calculated as an unsigned octet with the value:

RemStr or LocStr = (mas × 128) + 2 × nns + 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.

5.5.2.5 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 80.

```
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)
  }
```

7	6	5	4	3	2	1	0
rsv1	rsv2	rsv3	rsv4	inh	c12	dmb	dma

Figure 80 – Master_Report

5.5.2.6 Topo Counter

Each node shall implement the Topo Counter, which is a modulo 64 counter incremented by 1 or 2 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 81.

```
Topo_Counter::=         UNSIGNED6           -- 0 = not used
```

7	6	5	4	3	2	1	0
x	x	topo_counter					

Figure 81 – Format of Topo Counter

NOTE 1 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.

After a temporary power-down the node shall ensure to change the Topo Counter compared with the value before power-down.

NOTE 2 The Topo Counter may be stored in non-volatile memory (as it is used for quick reinsertion). When power is restored, the node is able to obtain its former Topo Counter and can change it.

After a redundancy-switchover the newly active node shall ensure to change the Topo Counter compared with the value of the previously active node.

NOTE 3 One node can use an even-numbered Topo Counter and the other node can use an odd-numbered Topo Counter. The Topo Counter is incremented by 2 for each successive Topography received by the node. This method ensures that the Topo Counter remains even- and odd-numbered resp. In case of redundancy-switchover the Topo Counter changes from even-numbered to odd-numbered or vice versa.

5.5.2.7 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 82.

```
Master_Topo::=          UNSIGNED12
```

11	10	9	8	7	6	5	4	3	2	1	0
master_topo											

Figure 82 – 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.

5.5.2.8 Inauguration_Counter

Each node shall implement a 16-bit Inauguration_Counter to register the number of times it went through an inauguration procedure, passing through the state of unnamed node.

NOTE This counter is used for diagnostics.

5.5.3 Detection of other compositions (informal)

5.5.3.1 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 (5.2.2.4).

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.

5.5.3.2 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 μs 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 μs 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 ± 4,0 ms around 25,0 ms to avoid repetitive collisions;

d) an End Node, after sending a Detect Request, shall ignore frames which are neither Detect Response nor Detect Request received during T_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 83 shows the timing diagram of a typical detection process.

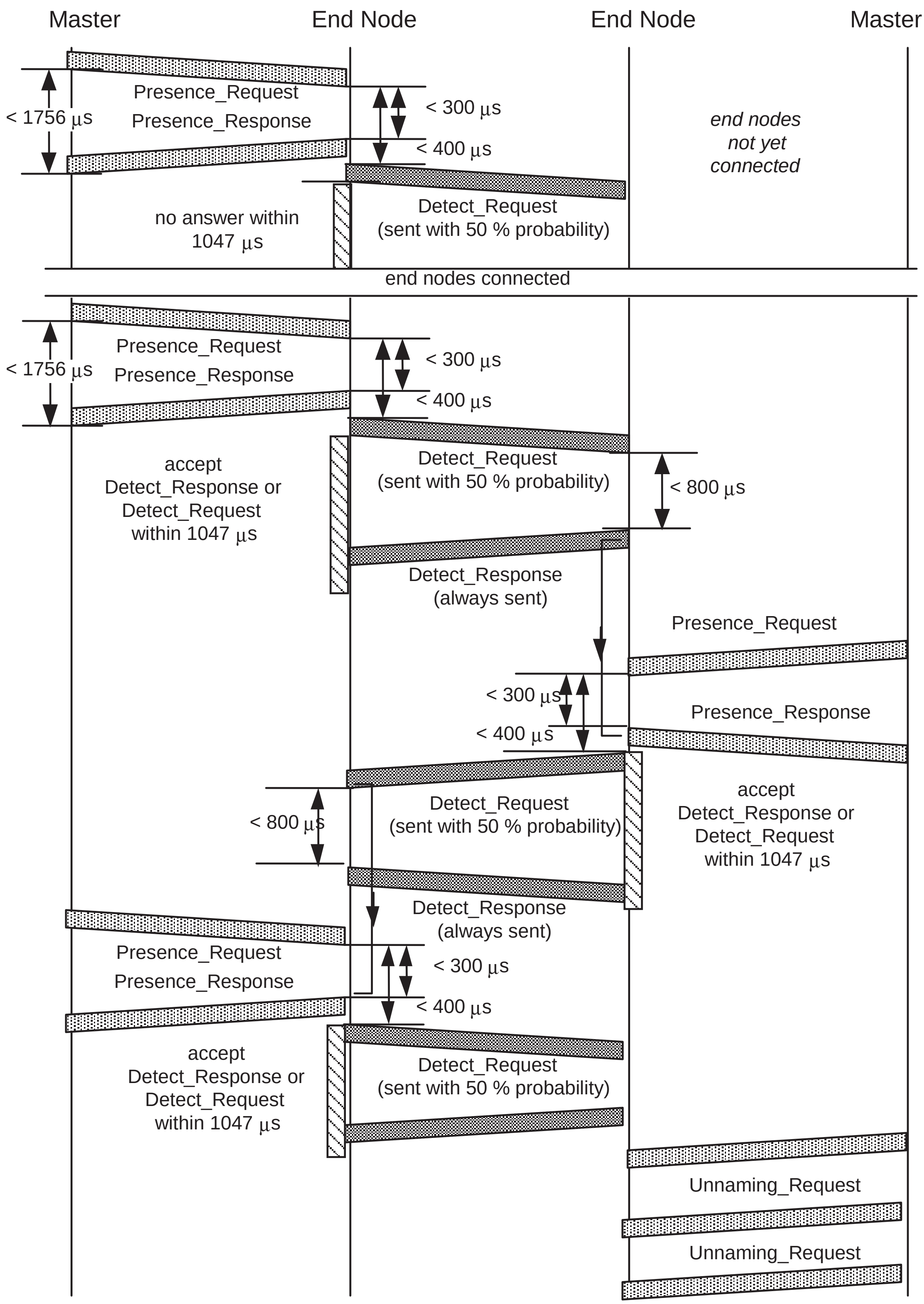


Figure 83 – Timing Diagram of detection protocol

5.5.4 State diagrams of the inauguration

5.5.4.1 Node structure

A node shall be in one of the major states shown in Figure 84. These major states subdivide into minor states that are defined in the following subclauses.

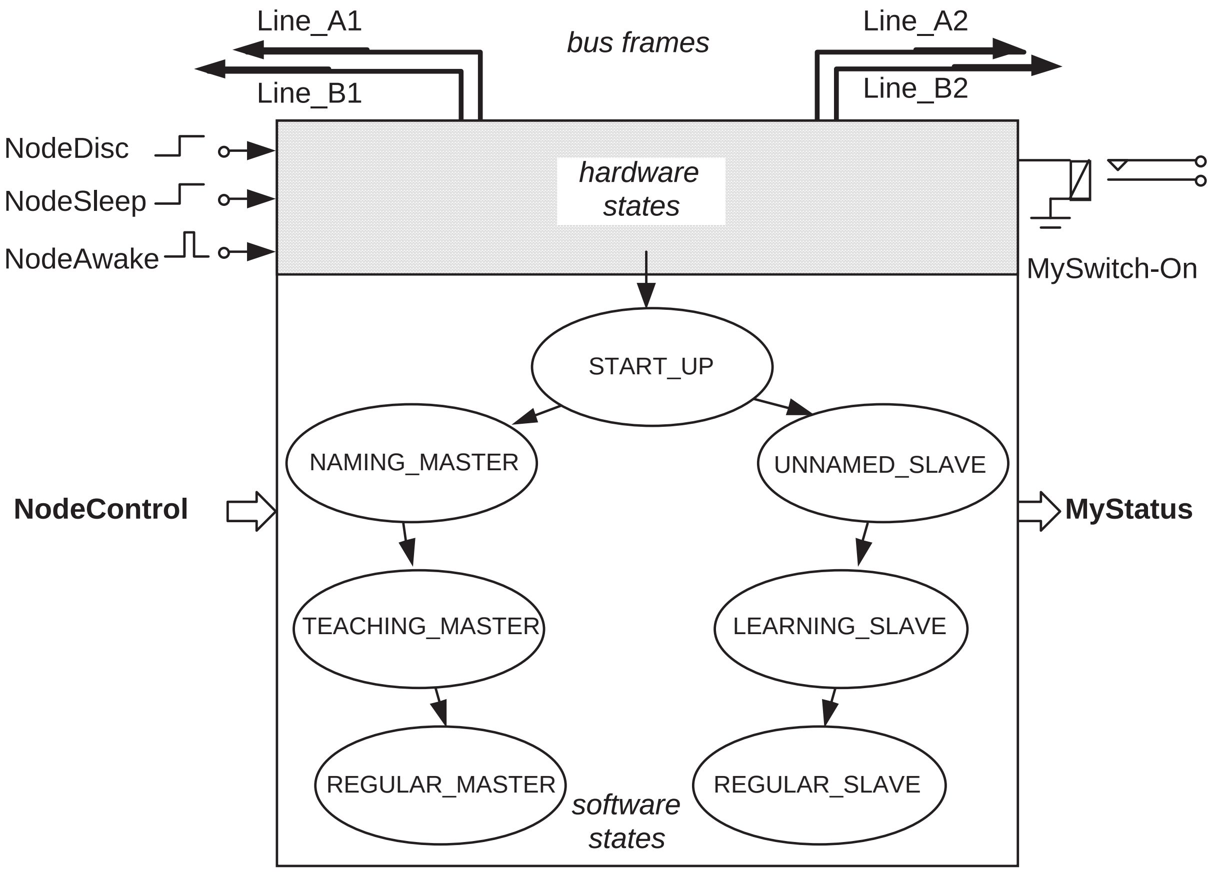


Figure 84 – Major node states and application settings

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.

5.5.4.2 Inauguration process structure

Conceptually, and for the purpose of this specification, the node activity is split into three processes, as shown in Figure 85:

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.

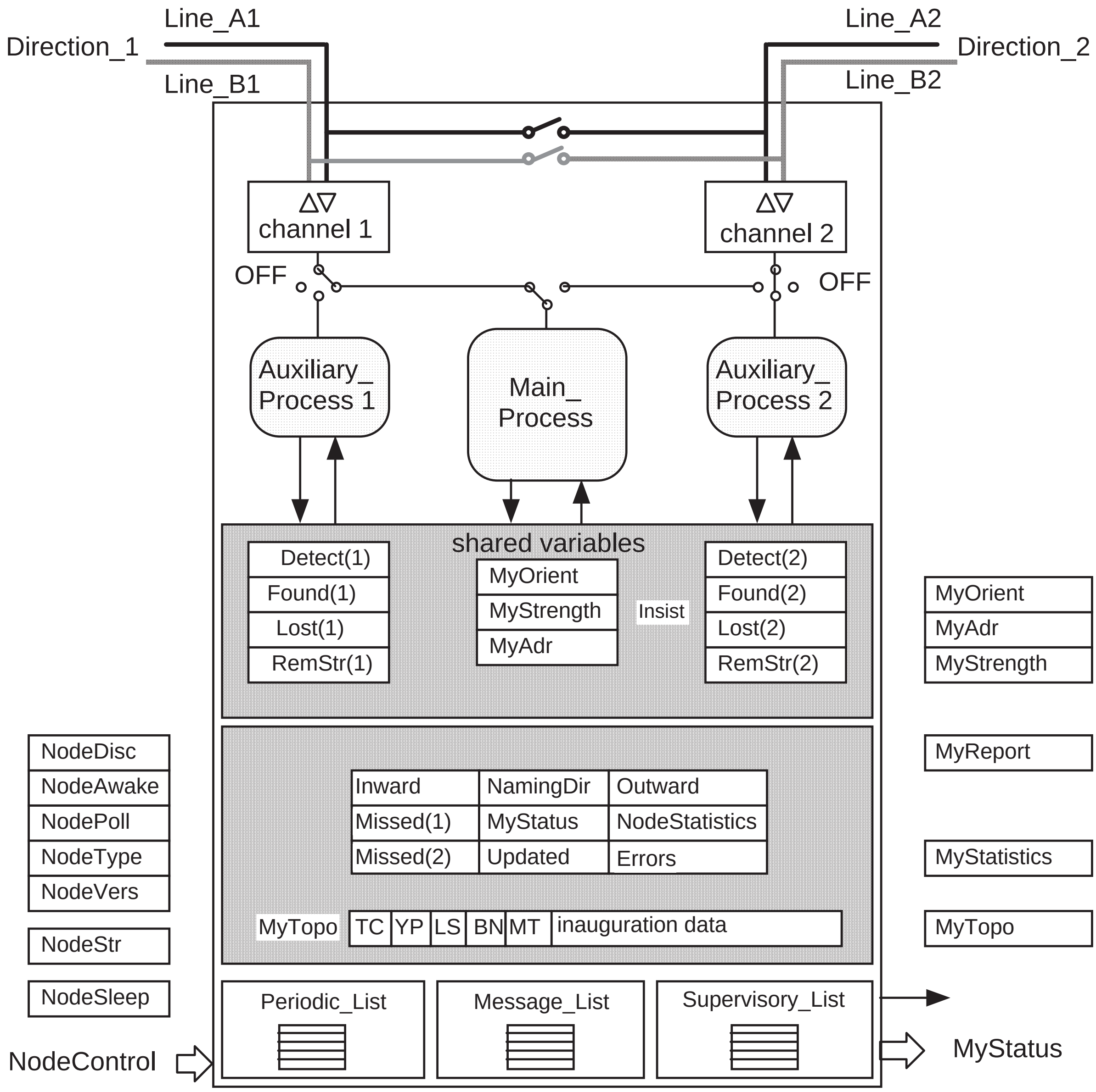


Figure 85 – Node processes (End Setting)

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):

a) by sending frames over the bus;

b) by starting time-outs;

c) by common (polled) variables.

The transitions within a process are triggered:

d) by control variables;

e) by frames received over the bus;

f) 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.

5.5.4.3 Specification language

The inauguration procedure is specified in the SDL/GR language [ITU-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.

5.5.4.4 Node variables

5.5.4.4.1 NodeControl

A node shall be controlled by the NodeControl data structure specified in Table 9.

Table 9 – 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 min

5.5.4.4.2 MyStatus

A node shall make its status available to the application through the MyStatus data structure specified in Table 10.

Table 10 – 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 are 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 (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

5.5.4.4.3 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 11 .

Table 11 – Shared Variables of a node

Variable	Type	Meaning
Detect (1) Detect (2)	BOOLEAN1 BOOLEAN1	‘1’ if detection is enabled in Direction_1 ‘0’ if detection is enabled in Direction_2
Found (1) Found(2)	BOOLEAN1 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
Lost(1) Lost(2)	INTEGER8 INTEGER8	counters which detect the loss of an already detected, but yet unnamed node
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) RemStr(2)	Composition_Strength Composition_Strength	strength of another composition either detected by the own Auxiliary_Process or received in the remote_strength field of a supervisory frame

5.5.4.4.4 Main Process Variables

The variables listed in Table 12 are used in the Main Process. Other local variables appear in the SDL diagrams of the corresponding macros or procedures.

Table 12 – Variables of Main Process

Variable	Type	Meaning
Topo	RECORD	Topography, with one entry per named node
TopoCounter	Topo_Counter	see 5.5.2.6
MasterTopo	Master_Topo	see 5.5.2.7
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) Missed(2)	UNSIGNED8 UNSIGNED8	Detects the loss of an end node in each direction
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 13.

Table 13 – 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 in 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 in which the transmission requests have been detected (signalled by the C_bit)

5.5.4.5 Auxiliary_Process

5.5.4.5.1 Process

As shown in Figure 86, 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_REQUESTS which is entered as long as the Auxiliary_Process is active.

5.5.4.5.2 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 and 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.

5.5.4.5.3 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 a 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.

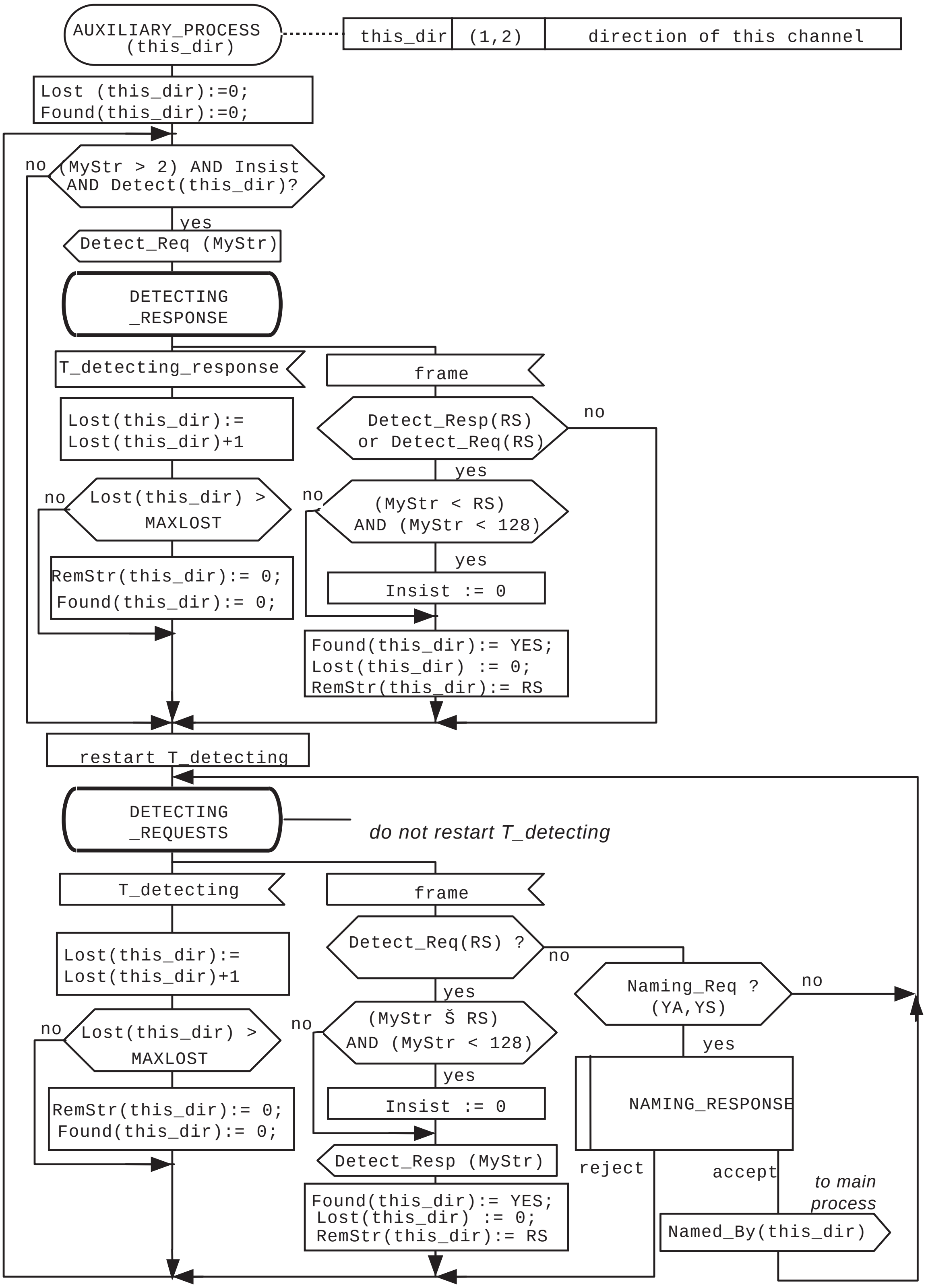


Figure 86 – AUXILIARY_PROCESS states

5.5.4.5.4 Macro 'NAMING_RESPONSE'

As shown in Figure 87, 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.

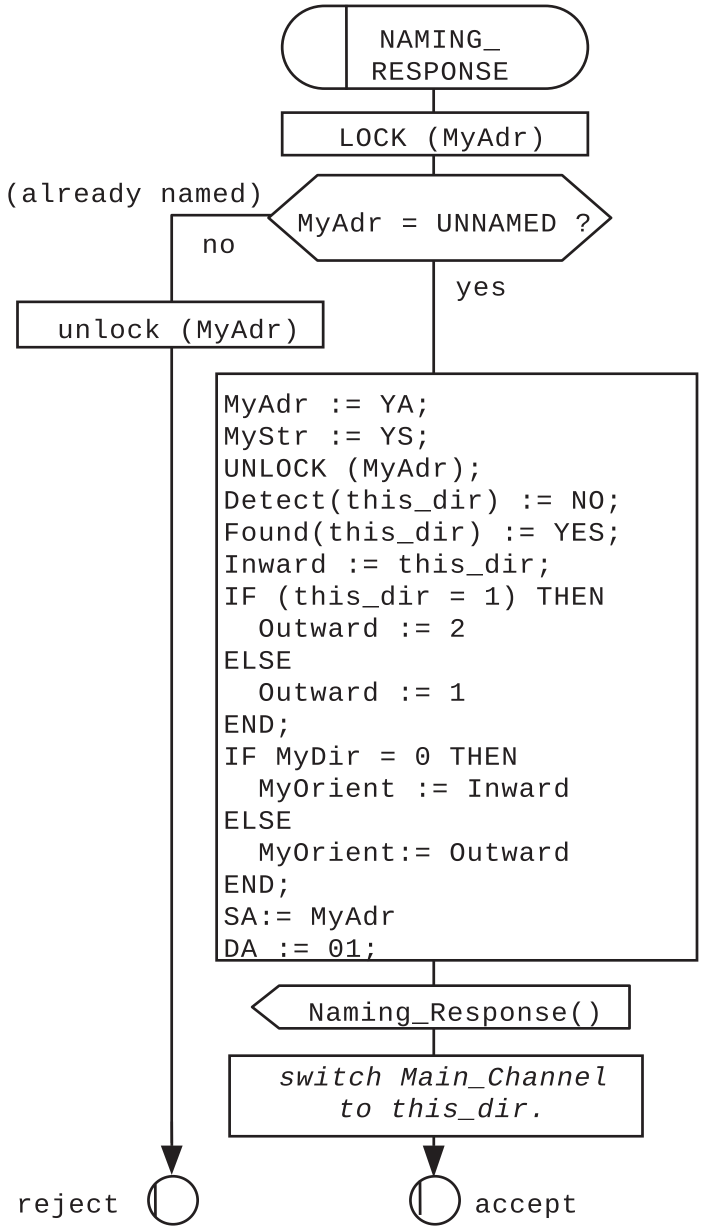


Figure 87 – NAMING_RESPONSE macro

5.5.4.6 Major states of the Main Process

As shown in Figure 88, Main Process is divided into 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.

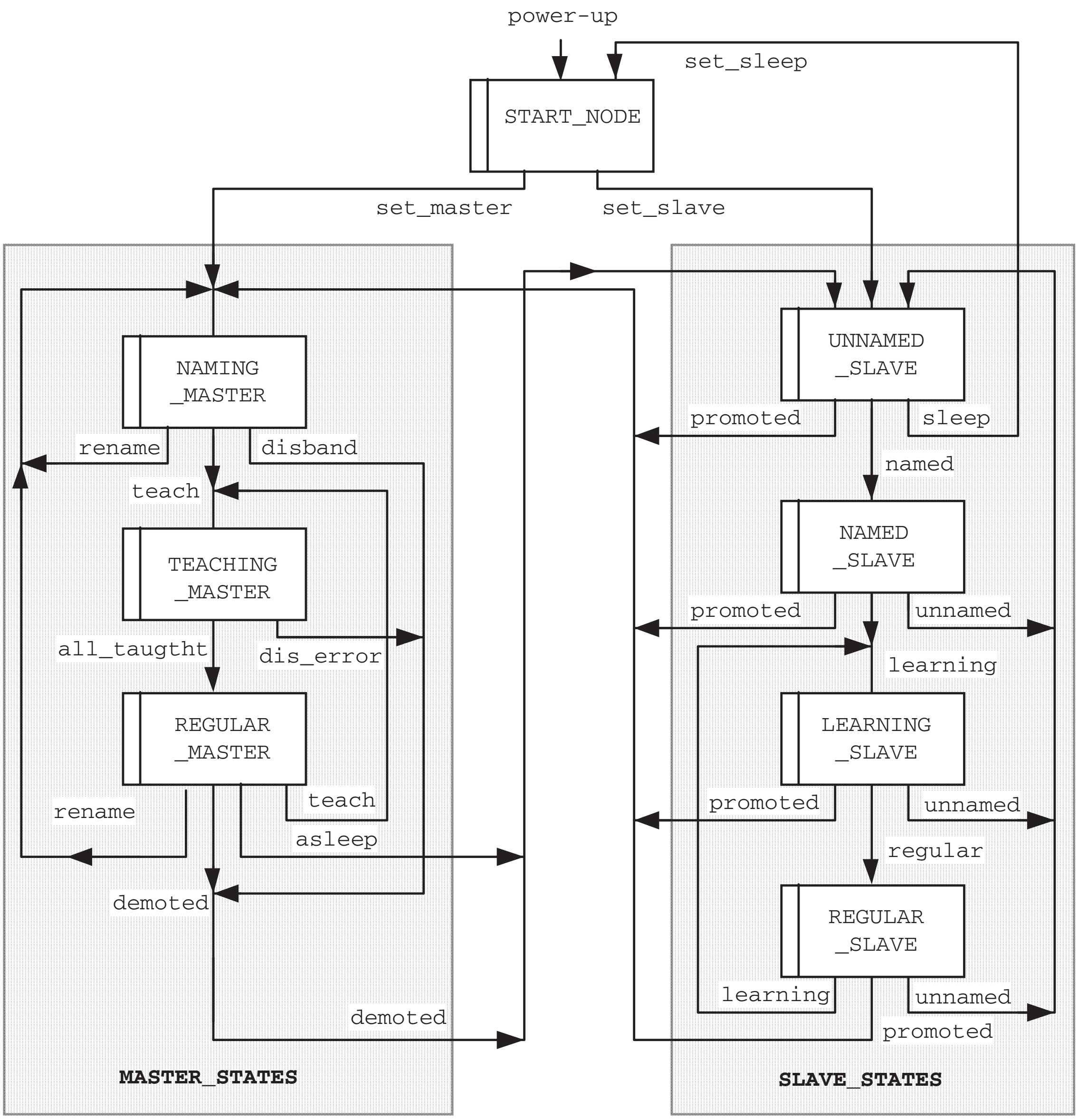


Figure 88 – States of MAIN PROCESS

The START_NODE state is described in Table 14.

Table 14 – ‘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 15.

Table 15 – ‘MASTER STATES’

‘NAMING_MASTER’	Initialises the node as lone master. Sets or lets 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 consist)
‘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 16.

Table 16 – ‘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.

5.5.4.7 Macro 'START_NODE'

As shown in Figure 89, this macro defines the hardware-controlled and the software-controlled states which a node passes through before it becomes master or slave.

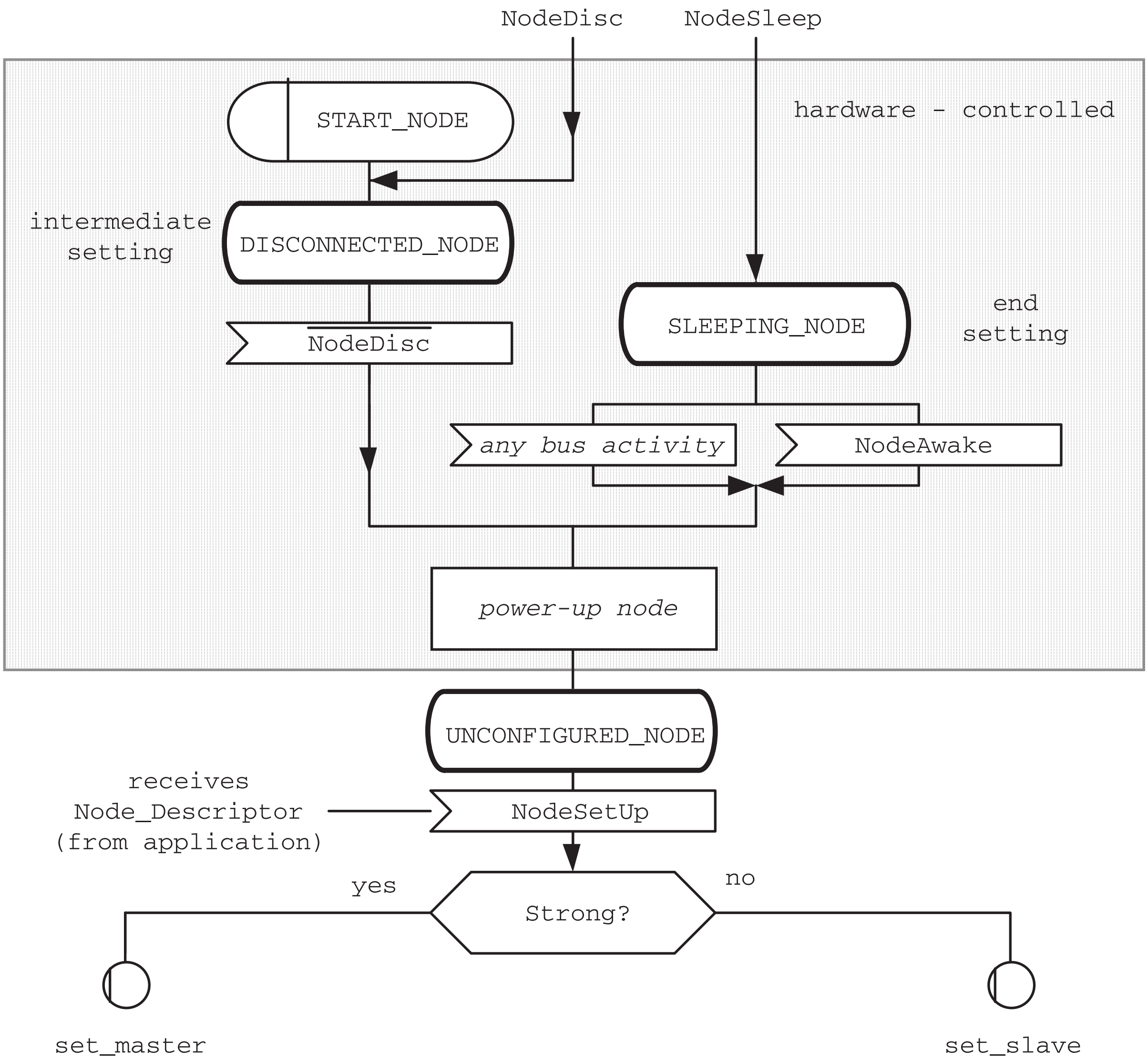


Figure 89 – Macro 'START_NODE'

5.5.4.7.1 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.

5.5.4.7.2 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 4.7.1.5), or any other means indicating that a well-formed frame has been received over either channel.

5.5.4.7.3 State 'UNCONFIGURED_NODE'

A node shall be in Intermediate 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.

5.5.4.8 Master States

5.5.4.8.1 Procedure REQUEST_RESPONSE

As shown in Figure 90, this procedure sends a request and waits for the corresponding response.

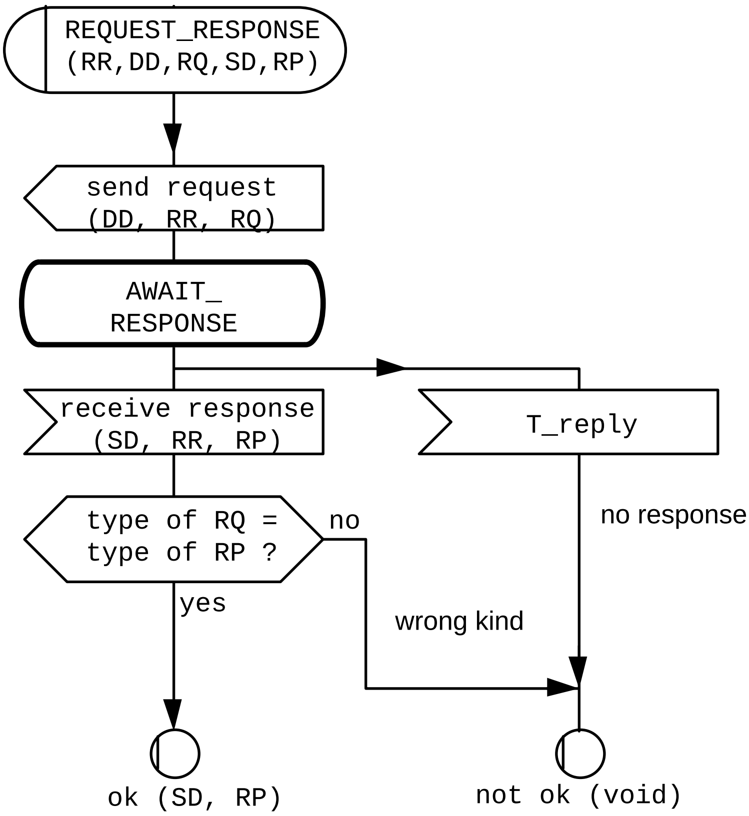


Figure 90 – Procedure REQUEST_RESPONSE

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.

5.5.4.8.2 Procedures 'SET_TO_END' and 'SET_TO_INT'

As shown in Figure 91, these two procedures set a node to End Setting, respectively to Intermediate Setting, if it is not already in that state.

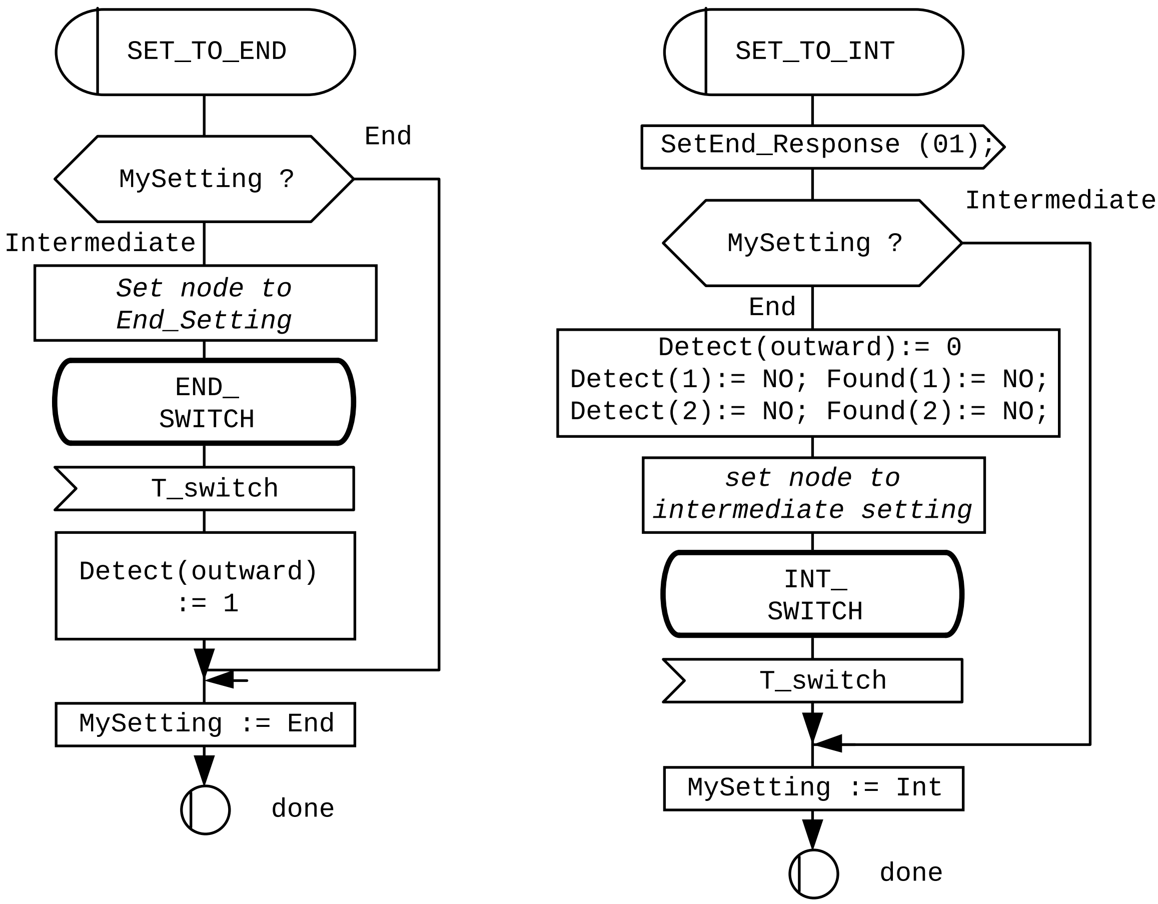


Figure 91 – Procedures ‘SET_TO_INT’ and ‘SET_TO_END’

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’.

5.5.4.8.3 Macro ‘INIT_MASTER’

As shown in Figure 92, 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 setup, the master is ready to name other nodes it may find.

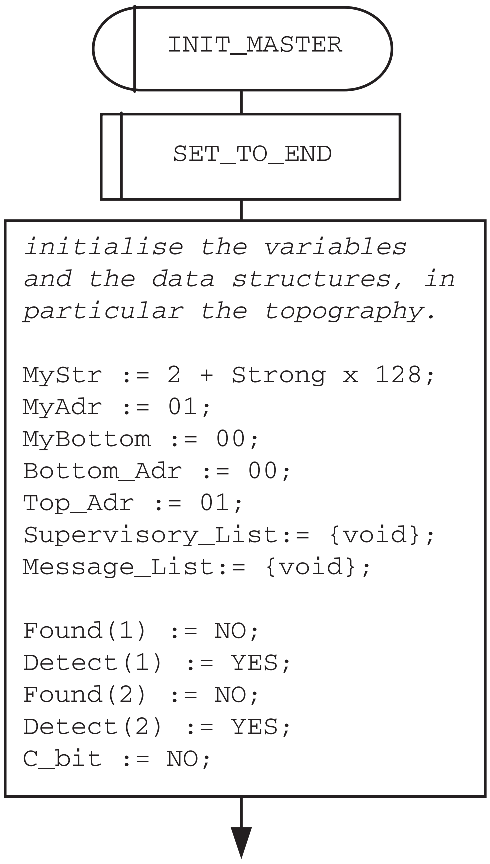


Figure 92 – Macro ‘INIT_MASTER’

5.5.4.8.4 Macro ‘NAMING_MASTER’

As shown in Figure 93, 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:

a) if the detected composition is stronger, the master shall disband the composition by going to the UNNAMING_MASTER macro;
b) 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);
c) 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;
d) if the end node reports an unnamed node, the master shall name the node as new end node;
e) 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.

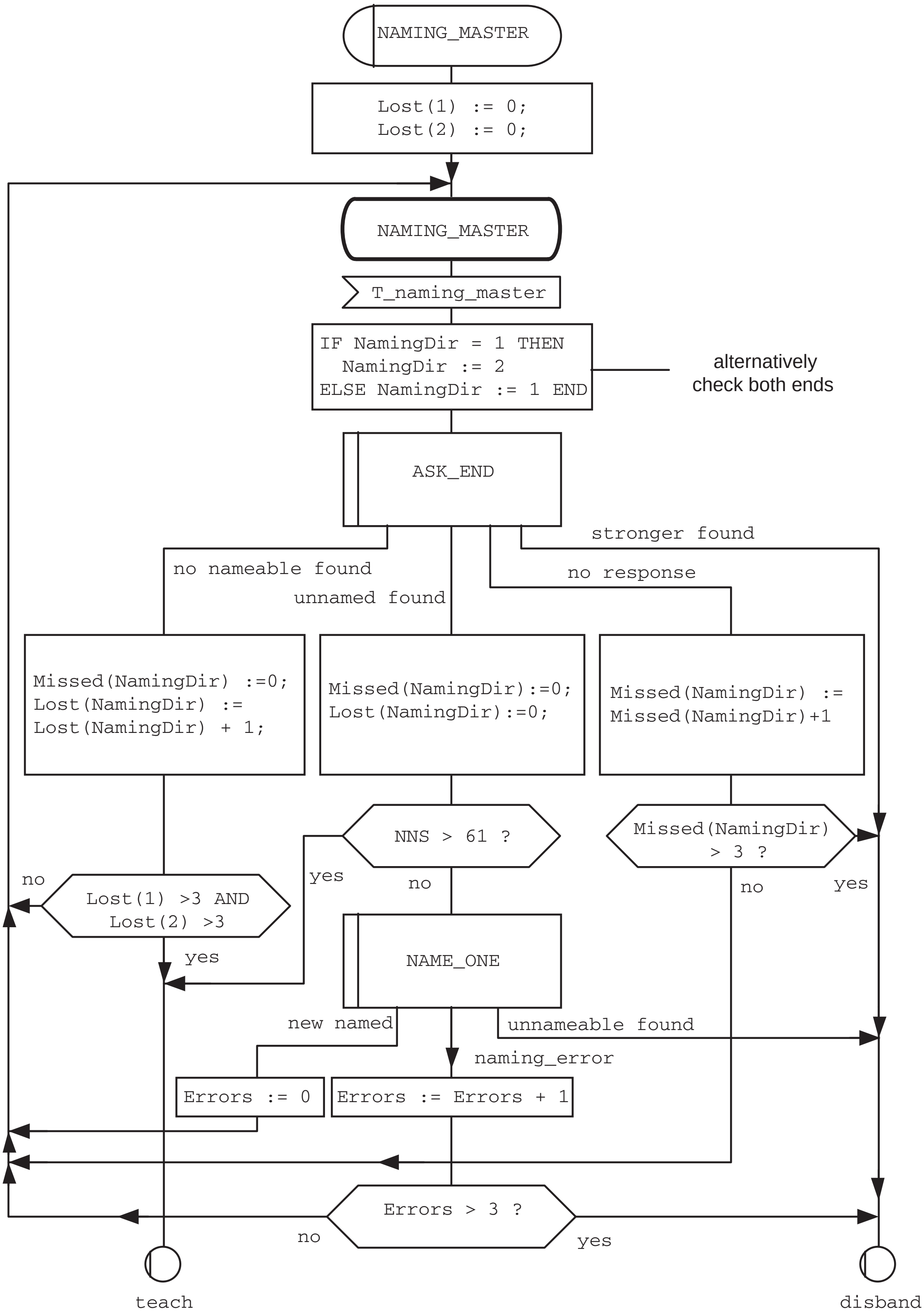


Figure 93 – Macro ‘NAMING_MASTER’

5.5.4.8.5 Macro 'ASK_END'

As shown in Figure 94, this macro checks the presence of the End Node in direction 'NamingDir' and requests 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:

a) no nameable node found (either no node, or an already named node);

b) unnamed found (the remote node would accept naming);

c) stronger found (the remote node belongs to a stronger composition).

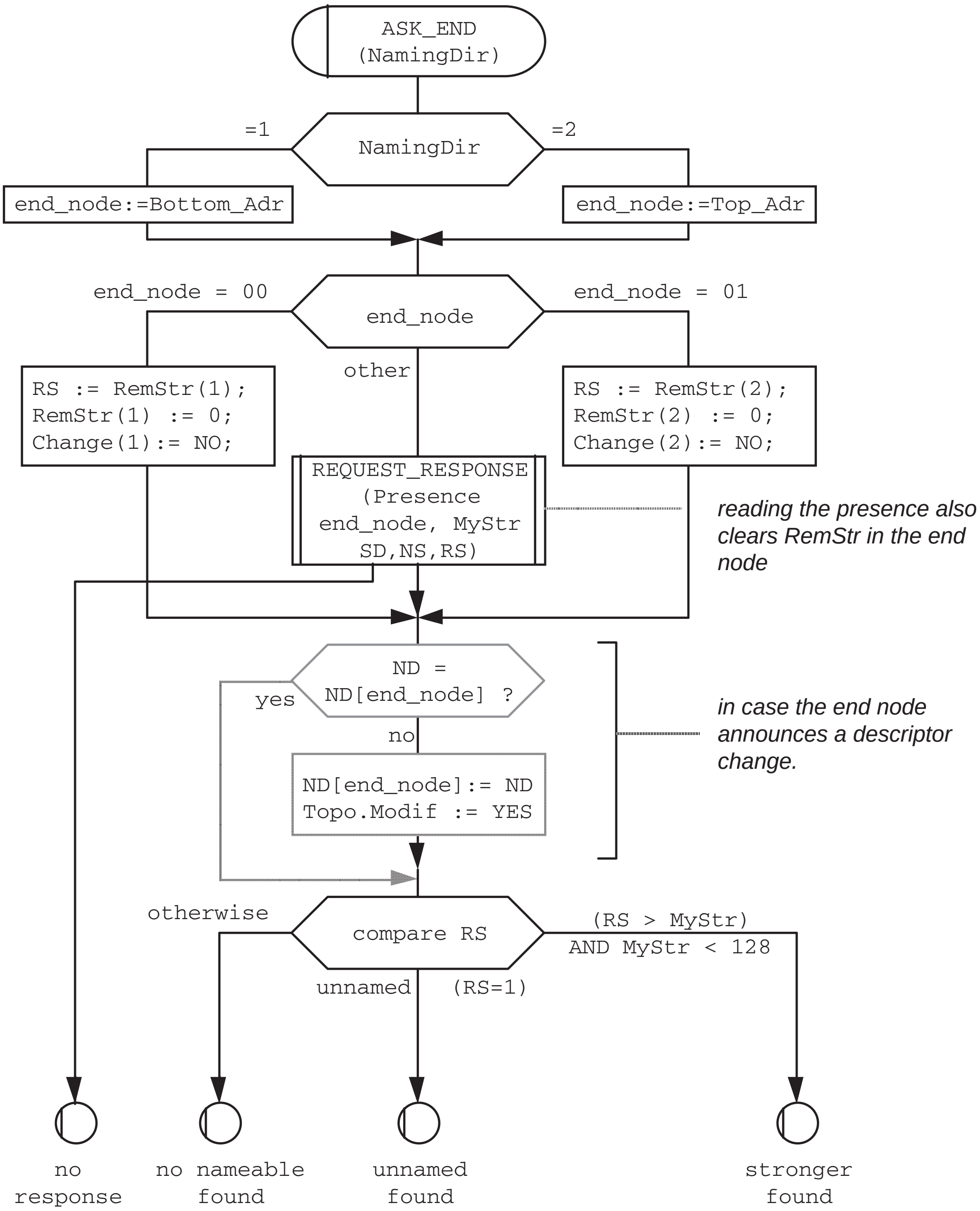


Figure 94 – Macro ASK_END

5.5.4.8.6 Procedure 'NAME_ONE'

As shown in Figure 95, 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 shall 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 update the topography, the addresses of the End Nodes and the composition strength and 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.

Variable	Type	Meaning
end_node	INT8	address of end node in either direction

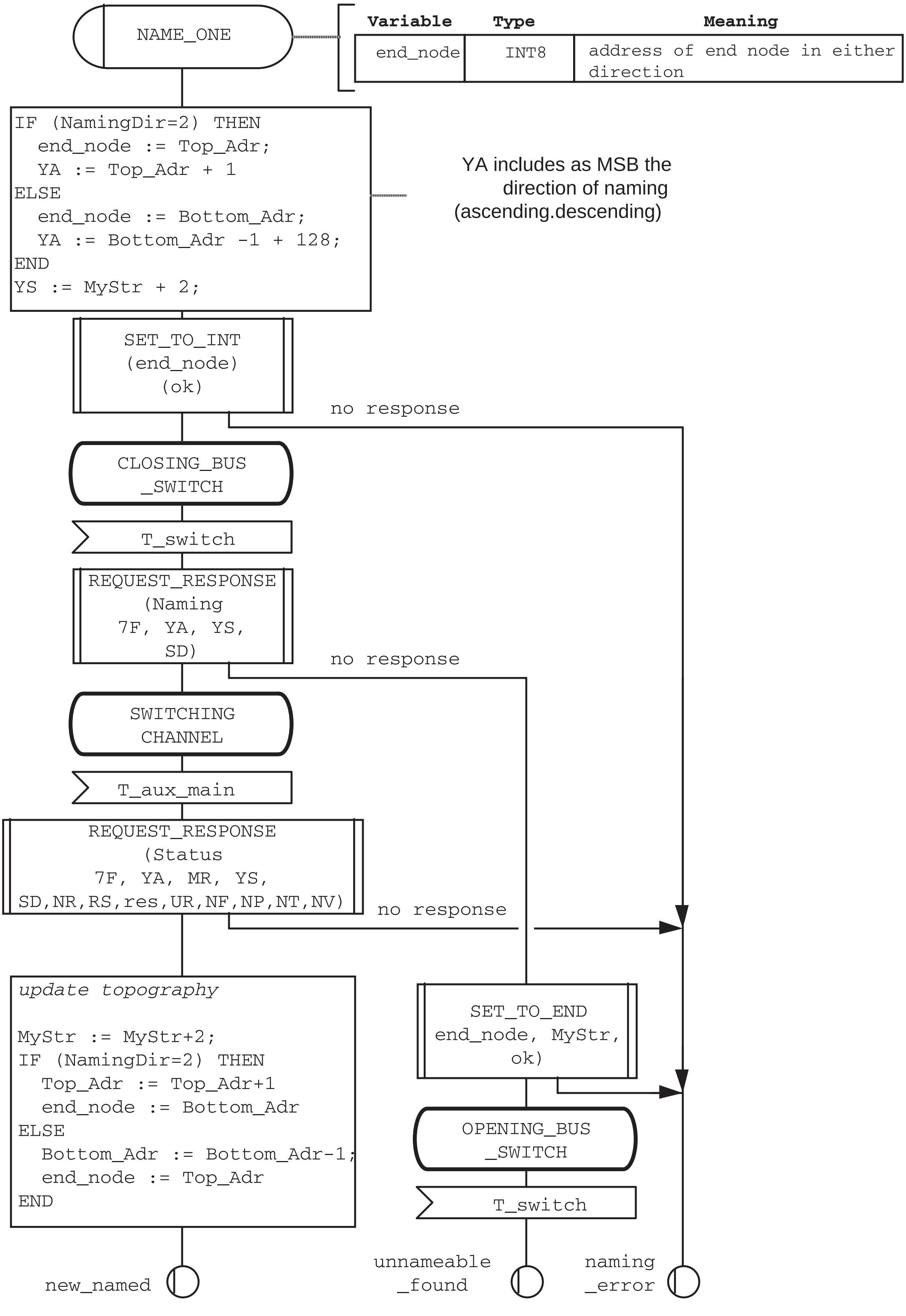


Figure 95 – Procedure NAME_ONE

5.5.4.8.7 Macro 'TEACHING_MASTER'

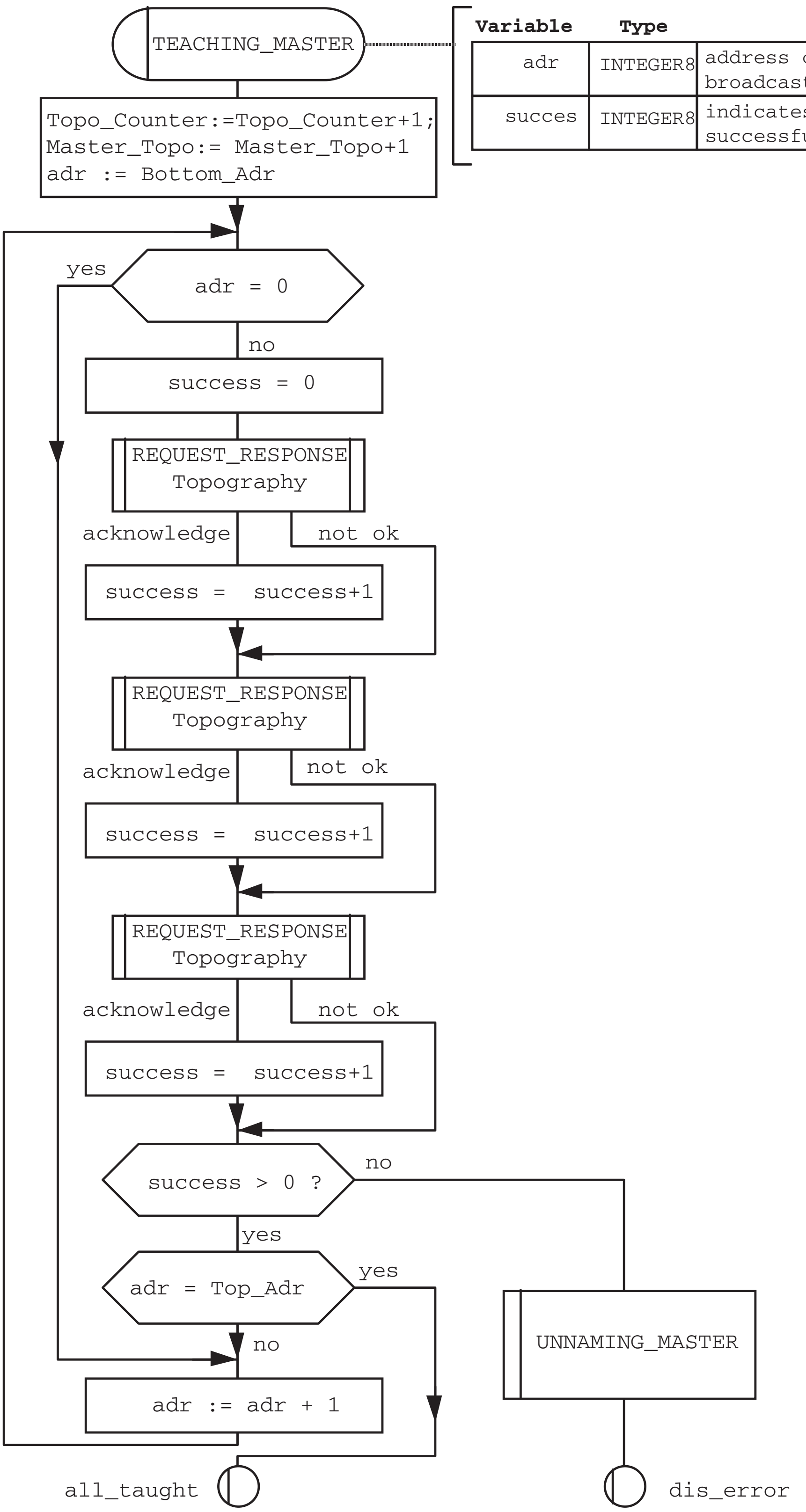
As shown in Figure 96, the master distributes the Topography information to all named nodes when it detects no more nodes to name or when it detects a node which signals a change in its Node Descriptor.

To this effect, it shall increment TopoCounter and MasterTopo.

The node shall send three times the 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.



Variable	**Type**	**Meaning**
adr	INTEGER8	address of the node asked to broadcast its Topography_Respon
succes	INTEGER8	indicates the number of successful distributions

Figure 96 – Macro TEACHING_MASTER

5.5.4.8.8 Macro 'UNNAMING_MASTER'

As shown in Figure 97, the master disbands its composition by unnaming 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.

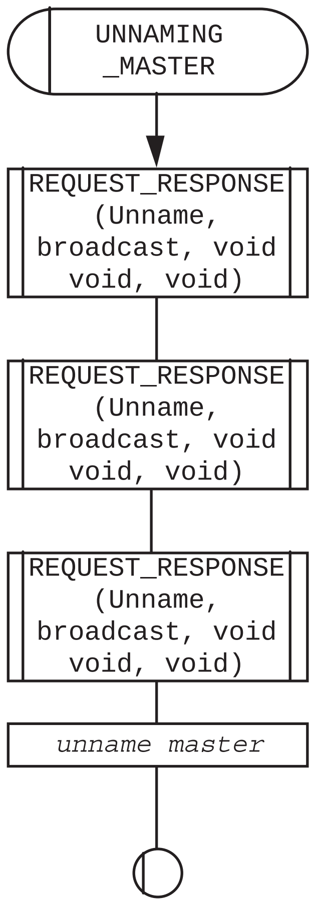


Figure 97 – Macro 'UNNAMING_MASTER'

5.5.4.8.9 Macro 'REGULAR_MASTER'

As shown in Figure 98, 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 5.5.4.8.11;

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 × 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;

The master shall leave the macro ‘REGULAR_MASTER’:

d) 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’;
e) 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’;
f) if it detects a composition change, and then go to the macro ‘TEACHING_MASTER’;
g) if it finds a composition stronger than its own with inauguration enabled and then disband its composition before going to UNNAMED_SLAVE;
h) 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.

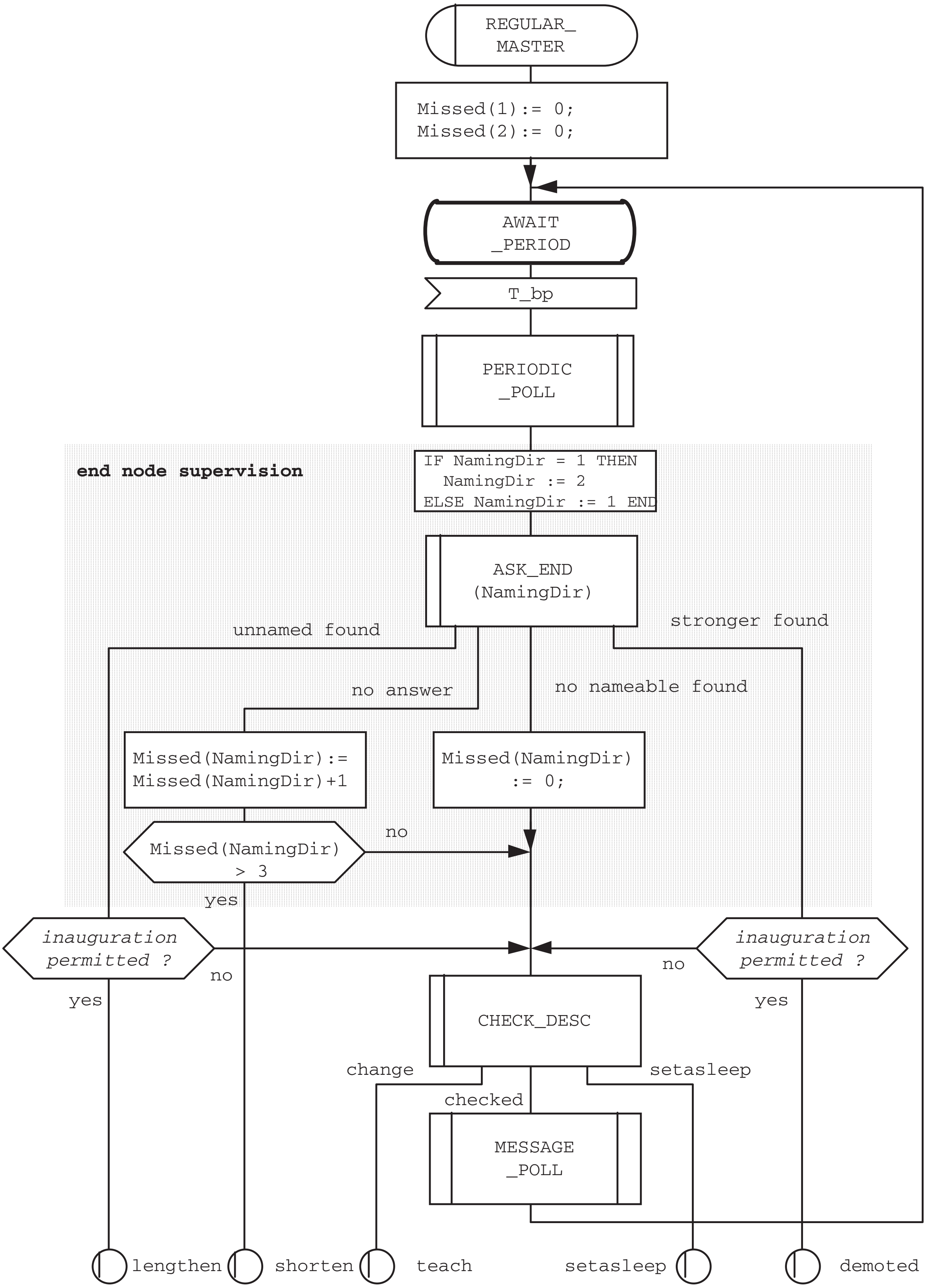


Figure 98 – Macro ‘REGULAR_MASTER’

5.5.4.8.10 Macro 'CHECK_DESC'

As shown in Figure 99, 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.

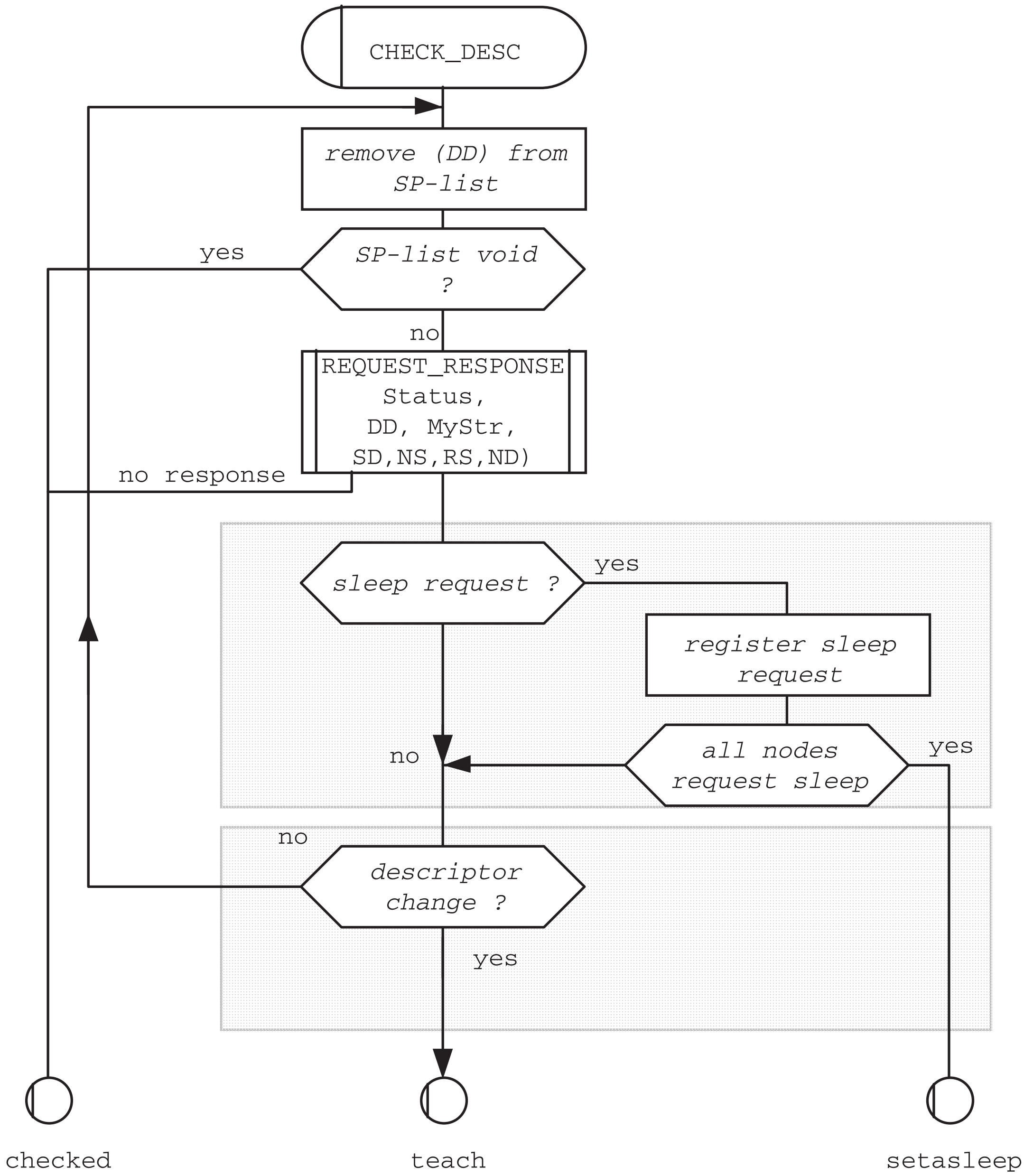


Figure 99 – Macro CHECK_DESC

5.5.4.8.11 Macro 'PERIODIC_POLL'

As shown in Figure 100, 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.

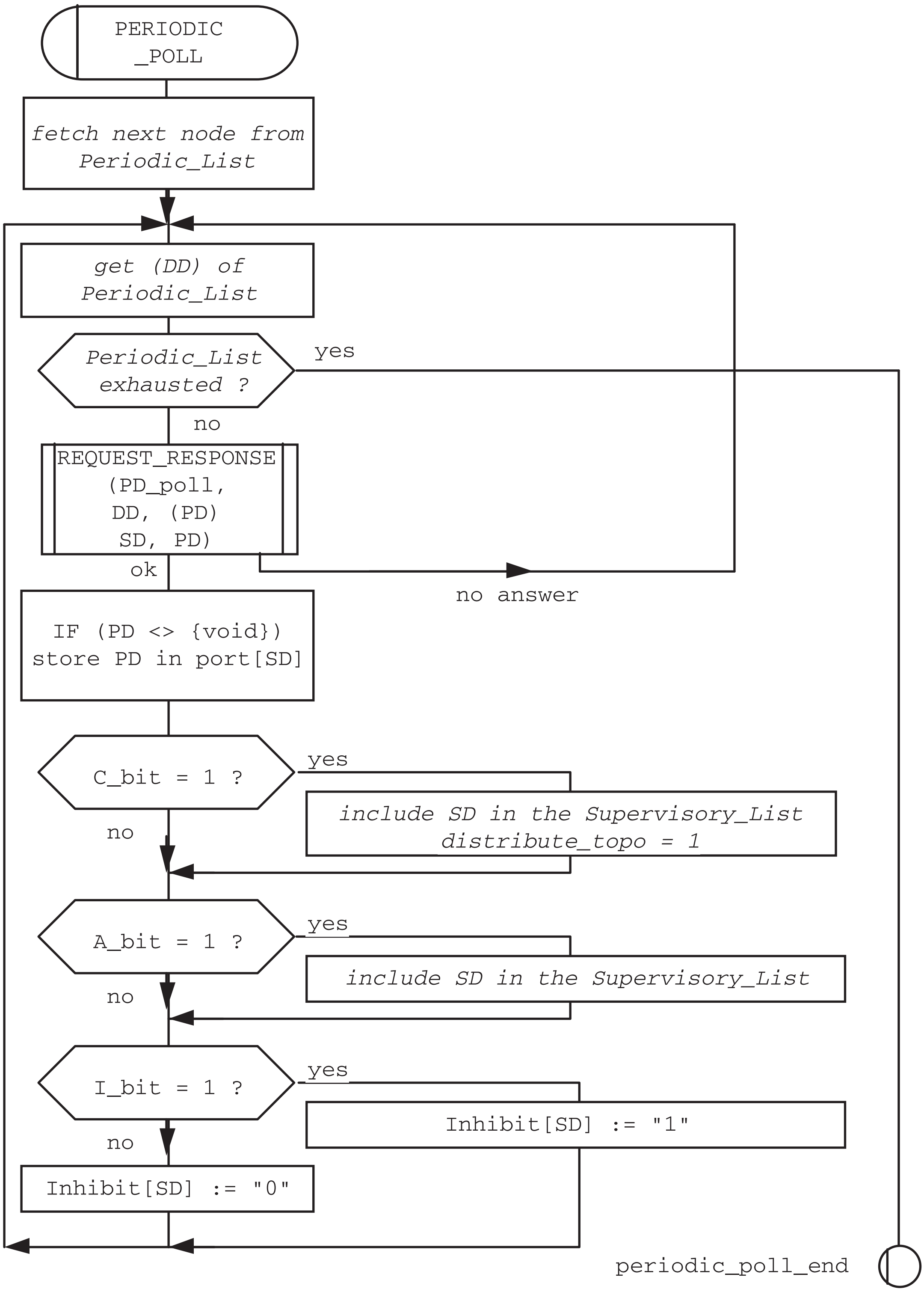


Figure 100 – Macro PERIODIC_POLL

5.5.4.8.12 Macro 'MESSAGE_POLL'

As shown in Figure 101, 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'.

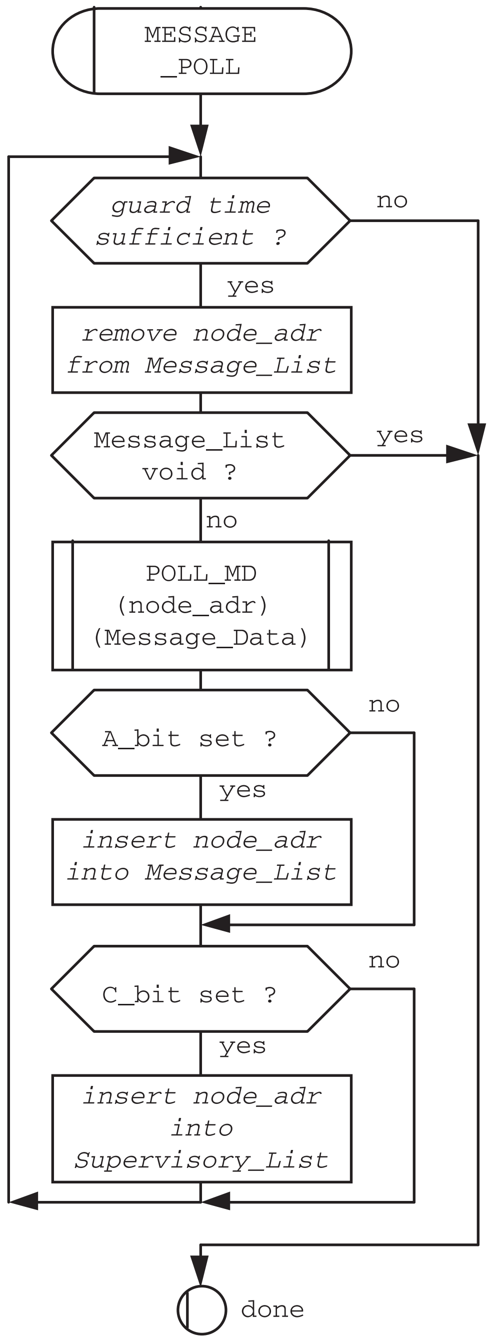


Figure 101 – Macro MESSAGE_POLL

5.5.4.9 Slave States

5.5.4.9.1 Macro 'UNNAMED_SLAVE'

As shown in Figure 102, 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 Auxilary channel,
   - locks its Main Channel towards the direction it has been named,
   - goes to the state NAMED_SLAVE.

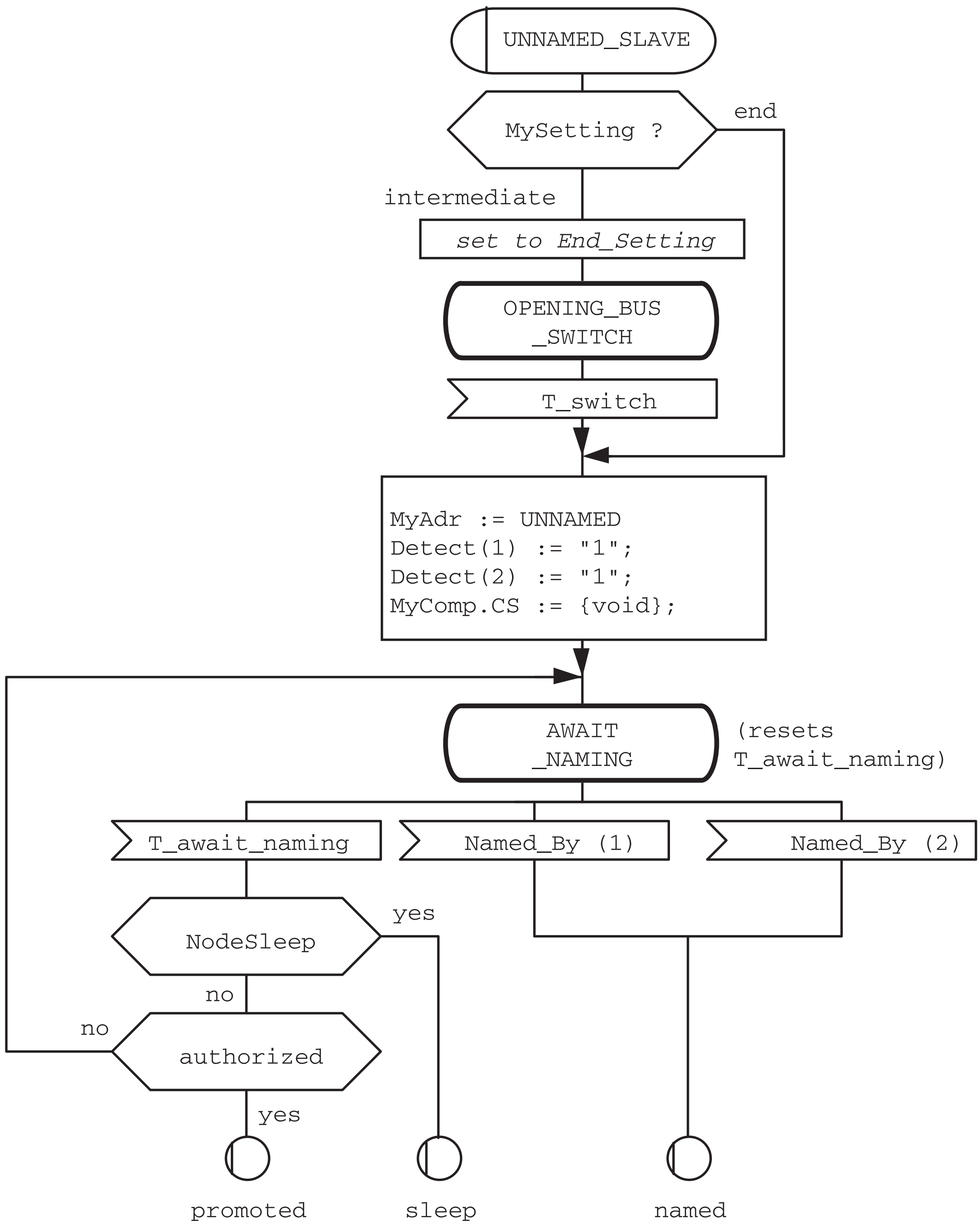


Figure 102 – States ‘UNNAMED_SLAVE’

5.5.4.9.2 Macro ‘NAMED_SLAVE’

As shown in Figure 103, 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_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_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, 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.

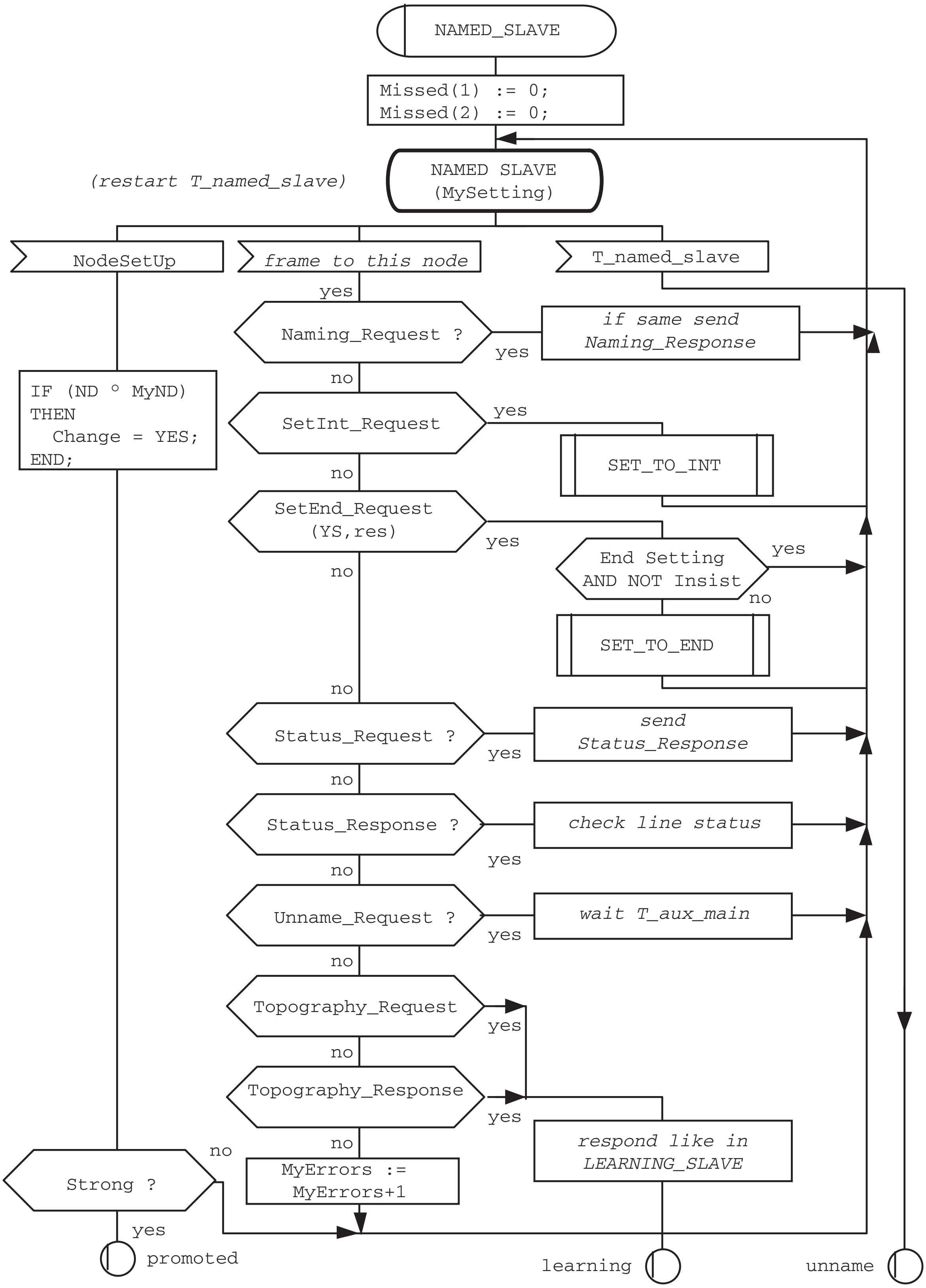


Figure 103 – States 'NAMED_SLAVE'

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.

5.5.4.9.3 Macro 'LEARNING_SLAVE'

As shown in Figure 104, 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.

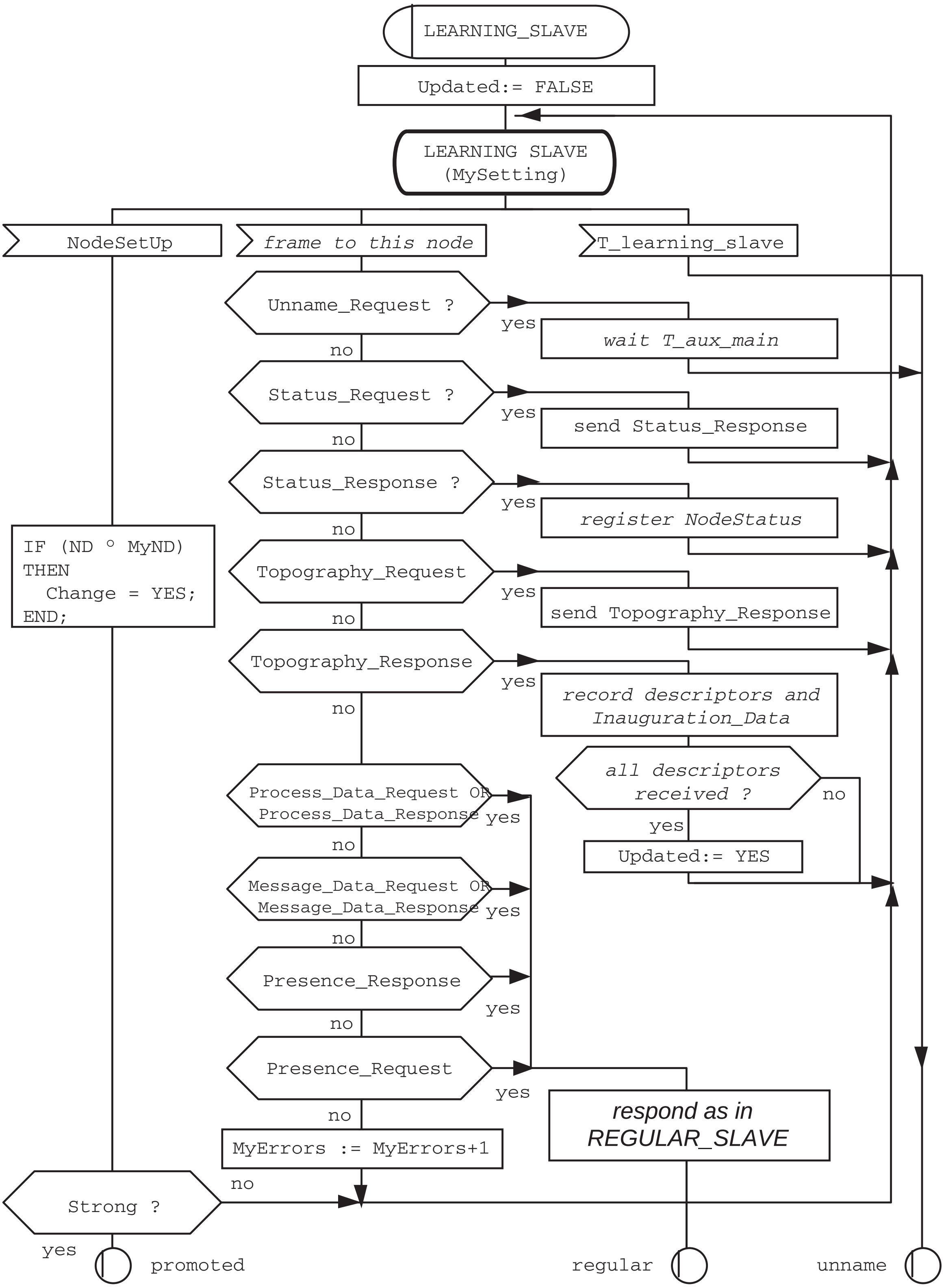


Figure 104 – Macro 'LEARNING_SLAVE'

NOTE In the learning phase, bus supervision is done with the Topography Responses, disregarding End Nodes.

5.5.4.9.4 Macro 'REGULAR_SLAVE'

As Figure 105 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 is not an End Node);

d) Presence Response:
   - restart the corresponsing 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 5.6.3);

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.

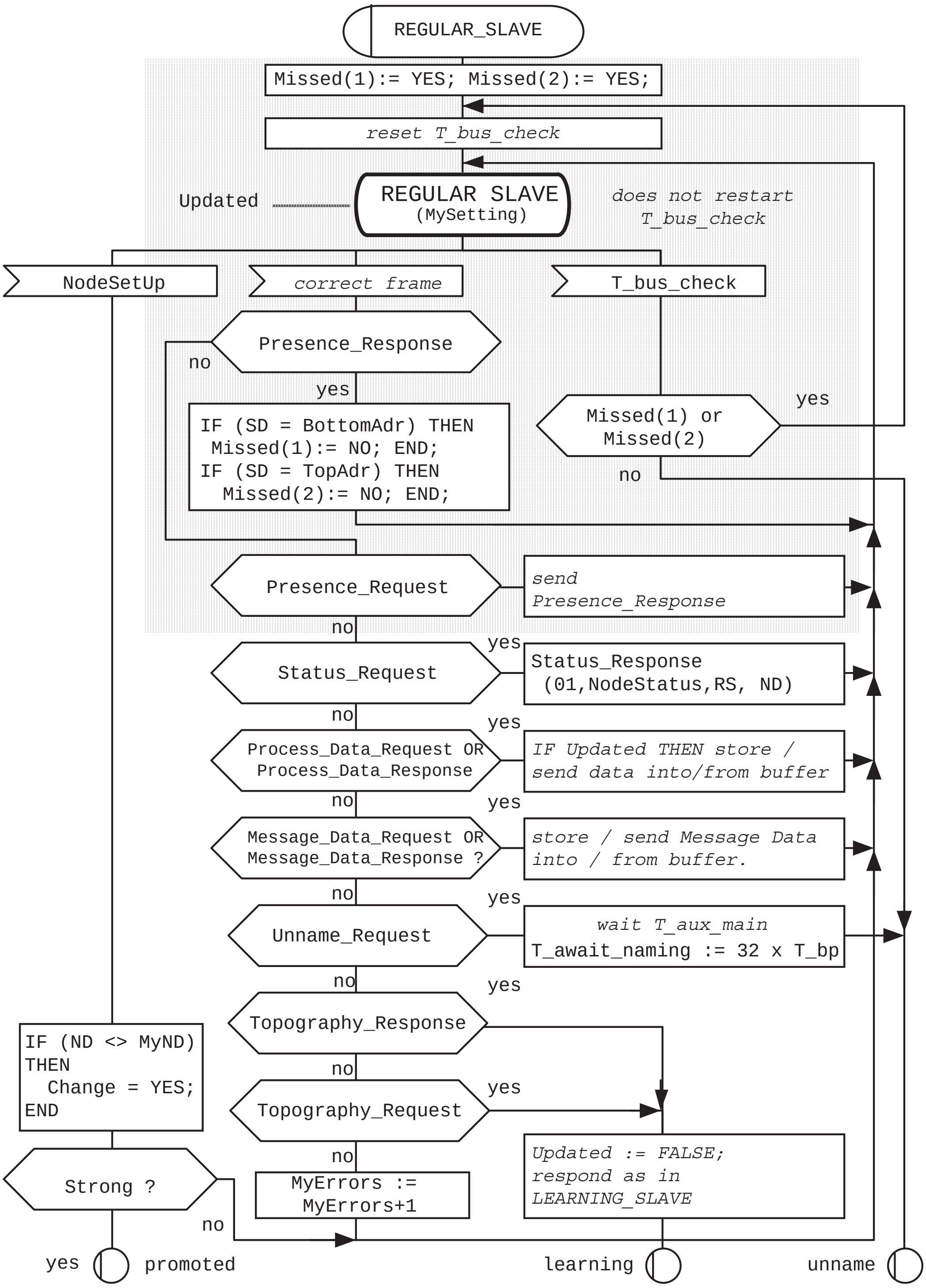


Figure 105 – Macro ‘REGULAR_SLAVE’

5.5.4.10 Time-outs

The recommended time-out values (±20 %) are listed in Table 17.

Table 17 – 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 – MyAdr) + 0,5) × T_bp	Nodes in direction 1 of the master
	3) (MyAdr-1) × T_bp	Nodes in direction 2 of the master
	4) T_await_max = 32 × 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 × T_bp	Supervision of the end nodes during regular operation
T_detecting	2 × T_naming_master	Interval between Detect_Requests (if any) during inauguration
	2 × 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 × T_bp	Minimum time between two consecutive inaugurations. n is set by the application
MAXLOST	50	T_detecting × 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

5.6 Link layer interface

5.6.1 Link layer layering

The Link Layer Interface provides three services, as shown in Figure 106:

a) the Link Process_Data_Interface (LPI), which is used by the Variables Service is specified in Clause 6 (Real-Time Protocols). Only the parameters specific to the WTB are specified in this standard;

b) the Link Message_Data_Interface (LMI), which is used by the Messages Service is specified in Clause 6 (Real-Time Protocols). Only the parameters specific to the WTB are specified in this standard;

c) 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.

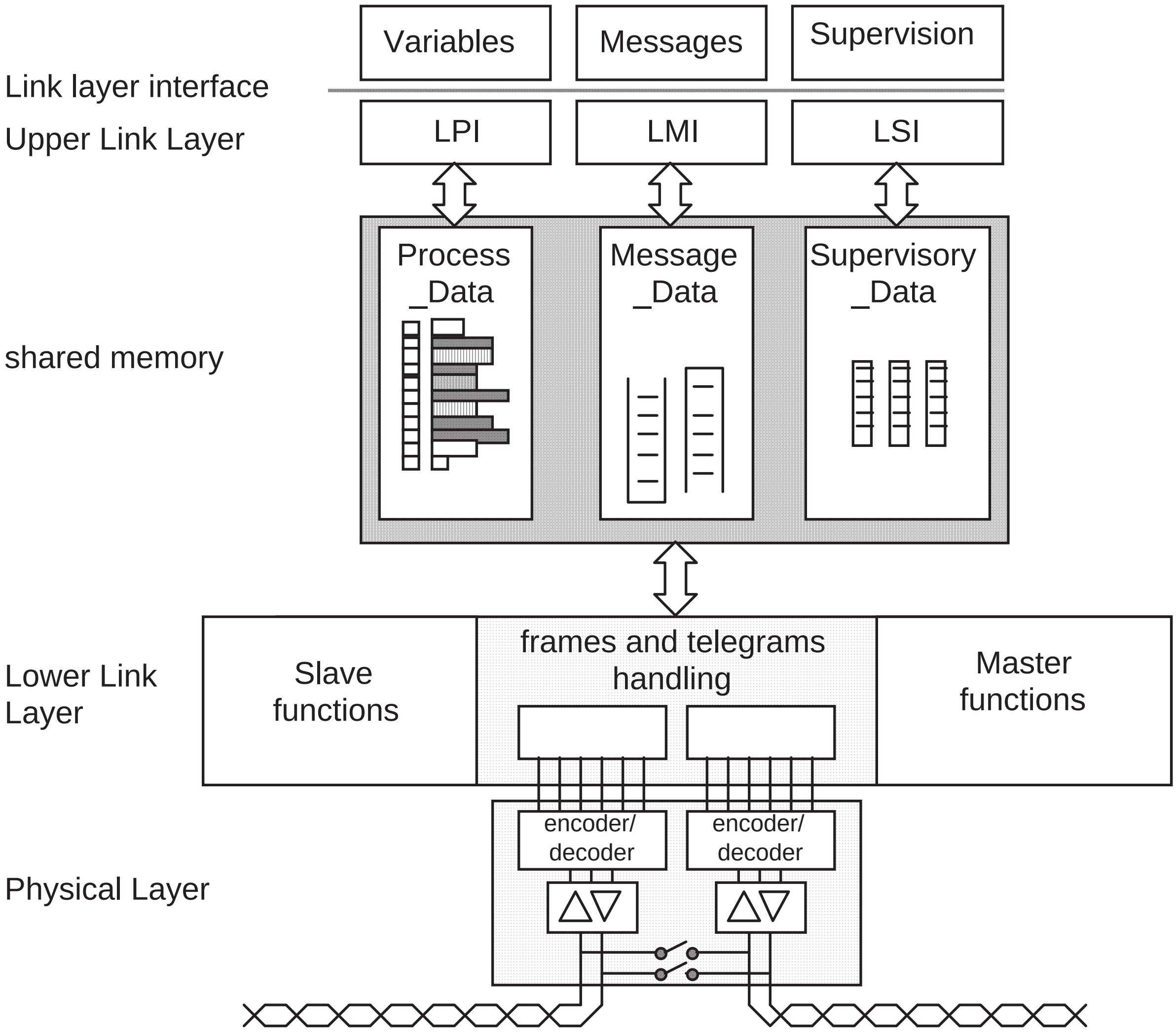


Figure 106 – Link layer layering

5.6.2 Link Process_Data_Interface

5.6.2.1 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 6 'Real-Time Protocols' specifies the Port access.

5.6.2.2 WTB specific

The Traffic_Store of the WTB shall hold up to 64 Ports, each with a maximum of 1 024 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 whose descriptor received in the Topography it cannot interpret.

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.

5.6.3 Link Message_Data_Interface

5.6.3.1 General

The Link Message_Data_Interface (LMI) specified in Clause 6 '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.

5.6.3.2 Packet size

The 'link_data_size' field in a void packet shall be zero.

The 'link_data_size' field shall be not larger than 128.

5.6.3.3 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).

5.6.3.4 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.

5.6.4 Link management interface

5.6.4.1 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 6) proposes a message format which is recommended as a parameter format.

5.6.4.2 Interface procedures

The procedures of this interface are prefixed by ls_t (link supervision, WTB).

5.6.4.2.1 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

5.6.4.2.2 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 TEACHING_MASTER
LS_REGULAR_SLAVE	6	node in state REGULAR_SLAVE or TEACHING_MASTER
LS_REGULAR_WEAK	7	node in state REGULAR_MASTER (weak) or TEACHING_MASTER

NOTE A node cannot indicate that it is in sleep state.

5.6.4.3 Reporting

5.6.4.3.1 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.

5.6.4.3.2 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_DISCONNEXION	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	state of a node changed
LR_POLL_LIST_OVF	34	partially operational
LR_ALLOWED	35	inauguration allowed

5.6.4.4 Initialisation service

5.6.4.4.1 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);`

5.6.4.5 Reset service

5.6.4.5.1 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);`

5.6.4.6 Configuration service

5.6.4.6.1 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.

5.6.4.6.2 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 5.5.2.1).

5.6.4.6.3 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 5.5.2.1).

5.6.4.6.4 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 5.6.4.6.3
poll_md_when_idle	UNSIGNED8	=1 if background scanning enabled, default: 0 (see 5.4.3.4).
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

5.6.4.6.5 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

The ‘node_descriptions’ shall be initialised before inauguration starts. At the end of the inauguration, the table ‘node_descriptions’ contains ‘nr_descriptors’ rows, one for each node.

5.6.4.7 Set Slave service

5.6.4.7.1 Procedure ls_t_SetSlave

Action	Prevents a node from becoming master.
Syntax	`LS_T_RESULT ls_t_SetSlave (void);`

5.6.4.8 Set Weak service

5.6.4.8.1 Procedure ls_t_SetWeak

Action	Enables a node to become weak master.
Syntax	`LS_T_RESULT ls_t_SetWeak (void);`

5.6.4.9 Set Strong service

5.6.4.9.1 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.

5.6.4.10 StartNaming service

5.6.4.10.1 Procedure ls_t_StartNaming

Action	Commands the node to start an inauguration.
Syntax	`LS_T_RESULT ls_t_StartNaming (void);`

5.6.4.11 Remove service

5.6.4.11.1 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);`

5.6.4.12 Inhibit service

5.6.4.12.1 Procedure ls_t_Inhibit

Action	Prevents a bus lengthening if additional nodes are detected.
Syntax	`LS_T_RESULT ls_t_Inhibit (void);`

5.6.4.13 Allow service

5.6.4.13.1 Procedure ls_t_Allow

Action	Enables a bus lengthening if additional nodes are detected.
Syntax	`LS_T_RESULT ls_t_Allow (void);`

5.6.4.14 SetSleep service

5.6.4.14.1 Procedure ls_t_SetSleep

Action	Causes the node to signal a sleep request.
Syntax	`LS_T_RESULT ls_t_SetSleep (void);`

5.6.4.15 CancelSleep service

5.6.4.15.1 Procedure ls_t_CancelSleep

Action	Causes the node to cancel a sleep request.
Syntax	`LS_T_RESULT ls_t_CancelSleep (void);`

5.6.4.16 GetStatus service

5.6.4.16.1 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.

5.6.4.16.2 Type_Node_Status

Attribute	Type	Meaning
node_report	BITSET8	‘C’ declaration corresponding to Node_Report (5.5.2.2)
user_report	BITSET8	‘C’ declaration corresponding to User_Report (5.5.2.3)

5.6.4.16.3 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

5.6.4.17 Get WTB nodes service

5.6.4.17.1 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

5.6.4.17.2 Procedure ls_t_GetWTBNodes

Action	Reads the list of Node Report and User Report of all nodes in the Topography.	
Syntax	`LS_T_RESULT ls_t_GetWTBNodes ( Type_NodeList * p_nodes );`	
Input	p_nodes	pointer to location where to put the list of nodes.

```
LS_T_RESULT          ls_t_GetWTBNodes
                     (
Type_NodeList *      p_nodes
                     );
```

5.6.4.18 Get Topography service

5.6.4.18.1 Procedure ls_t_GetTopography

Action	Allows the application to read the Topography distributed before regular operation begins.	
Syntax	`LS_T_RESULT ls_t_GetTopography ( Type_Topography * p_topography );`	
Input	p_topography	pointer to the place where to put the Topography.
Result		returns L_CONFIGURATION_INVALID if the Topography is not valid.

```
LS_T_RESULT          ls_t_GetTopography
                     (
Type_Topography *    p_topography
                     );
```

5.6.4.18.2 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

5.6.4.19 Change Node Descriptor service

5.6.4.19.1 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

```
LS_T_RESULT              ls_t_ChgNodeDesc
                         (
Type_NodeDescriptor *    node_descriptor
                         );
```

5.6.4.20 Change User Report service

5.6.4.20.1 Procedure ls_t_ChgUserReport

Action	Allows the application to modify User Report.	
Syntax	`LS_T_RESULT ls_t_ChgUserReport ( UNSIGNED8 set_mask UNSIGNED8 clear_mask );`	
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

```
LS_T_RESULT           ls_t_ChgUserReport
                      (
    UNSIGNED8         set_mask
    UNSIGNED8         clear_mask
                      );
```

5.6.4.21 Change Inauguration_Data service

5.6.4.21.1 Procedure ls_t_ChgInauguration_Data

Action	Allows the application to modify this node's Inauguration Data	
Syntax	`LS_T_RESULT ls_t_ChgInauguration_Data ( UNSIGNED8 inaug_data_size void* p_inauguration );`	
Input	inaug_data_size	size in octets of the inauguration data (≤ 124)
	p_inauguration	user-defined inauguration data

```
LS_T_RESULT           ls_t_ChgInauguration_Data
                      (
    UNSIGNED8         inaug_data_size
    void*             p_inauguration
                      );
```

5.6.4.22 Get Statistics service

5.6.4.22.1 Procedure ls_t_GetStatistics

Action	Provides statistical information about usage and errors	

Syntax	`LS_T_RESULT ls_t_GetStatistics (`
	`Type_LLStatisticData * p_statistic_data`
	`);`	

Input	p_statistic_data	pointer to the statistic data structure (see 5.6.4.22.3)

5.6.4.22.2 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.

5.6.4.22.3 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.

5.6.4.23 Get Inauguration_Data service

Action	Return pointer to inauguration data	

Syntax	`LS_T_RESULT`
	
	`void * *`
		`ls_t_GetInaugData`
		`(`
		`p_inaug_data_list`
		`);`

Output	p_inaug_data_list	pointer to inauguration_data of all named nodes

6 Real-Time protocols

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

6.1 General

6.1.1 Contents of this clause

This clause specifies one component of the Train Communication Network, the Real-Time Protocols, which provides communication between applications, within consists and between consists, as illustrated in Figure 107.

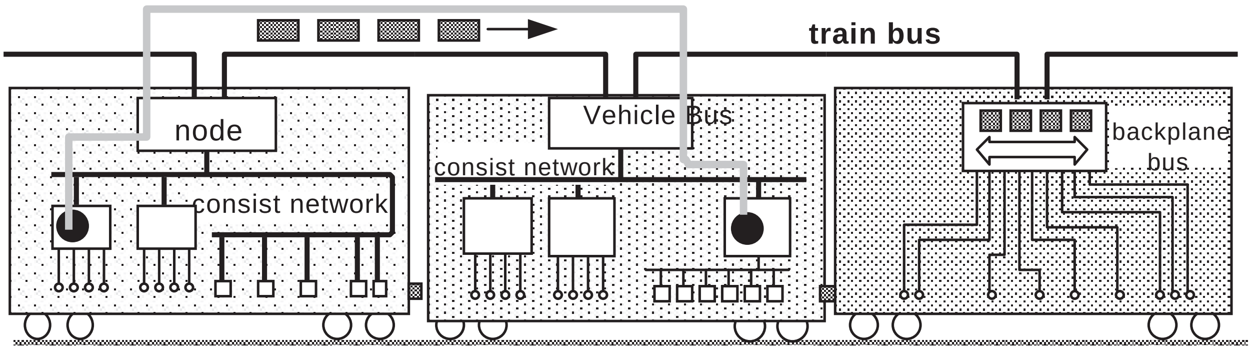


Figure 107 – Structure of the Train Communication Network

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).

6.1.2 Structure of this clause

This clause is structured similarly to the OSI communication model as shown in Figure 108.

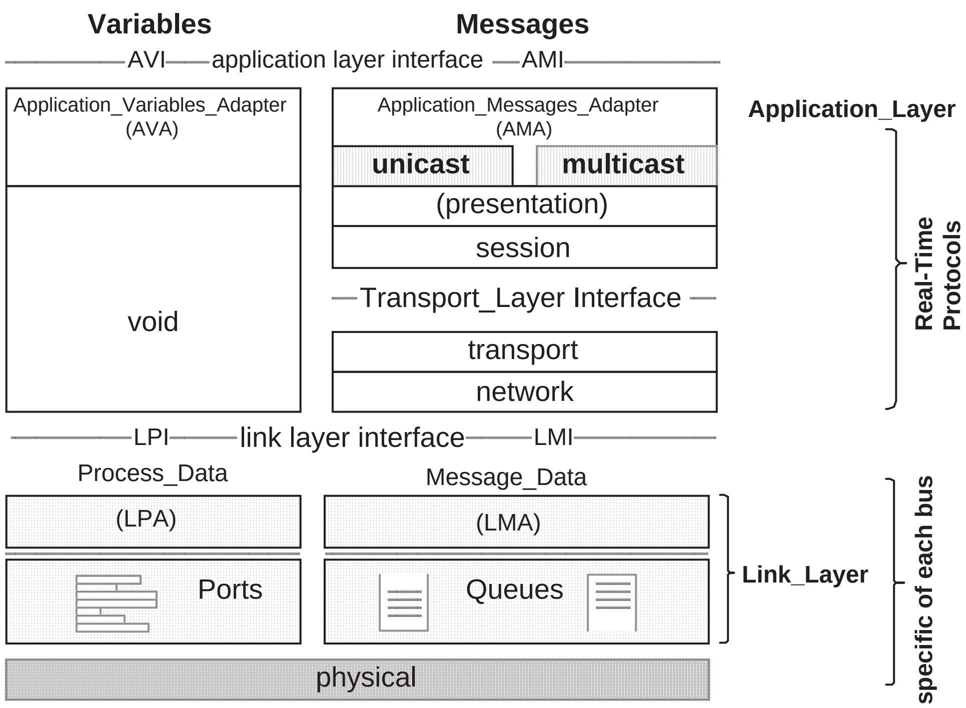


Figure 108 – Real-Time Protocols layering

Subclause 6.1 General

Normative requirements and definitions

Subclause 6.2 Variables – Services and Protocols

Process_Variable Objects

Dataset Interface

Application Interface for individual, Set and Cluster access

Subclause 6.3 Messages Services and Protocols

Architecture

Link layer_Interface

Network layer

Transport layer

Session layer

Application Interface

Subclause 6.4 Presentation and encoding of transmitted and stored data

6.2 Variables – Services and Protocols

6.2.1 General

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.

6.2.2 Link layer Interface for Process_Data

6.2.2.1 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.

NOTE Individual Process_Variables are not visible at the LPI level.

6.2.2.2 Datasets

6.2.2.2.1 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.

6.2.2.2.2 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.

6.2.2.2.3 Error handling

Undefined fields in a Dataset shall be overwritten with ‘1’.

If the link layer cannot guarantee consistency of a Dataset, for example 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.

6.2.2.2.4 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.

6.2.2.2.5 Synchronisation dataset

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

6.2.2.2.6 Dataset polling

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

6.2.2.2.7 Dataset identifier (DS_Name)

6.2.2.2.7.1 Dataset, port and Logical Address

Within a device, a Dataset shall be identified by 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.

6.2.2.2.7.2 DS_Name format

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

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

```
typedef struct                /* big-endian representation
*/
  {
  unsigned traffic_store_id:4,  /* DS_NAME first part */
  unsigned port_address    :12, /* DS_NAME second part */
  } DS_NAME;
```

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

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

6.2.2.2.7.3 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.

6.2.2.2.8 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.

6.2.2.3 Link_Process_Data_Interface primitives

6.2.2.3.1 General

The Link_Process_Data_Interface (LPI) shall provide for Dataset access the primitives illustrated in Figure 109 and listed in Table 18, as specified in the following subclauses.

The following subclauses do not imply a particular implementation. Any interface which provides the same semantics is allowed.

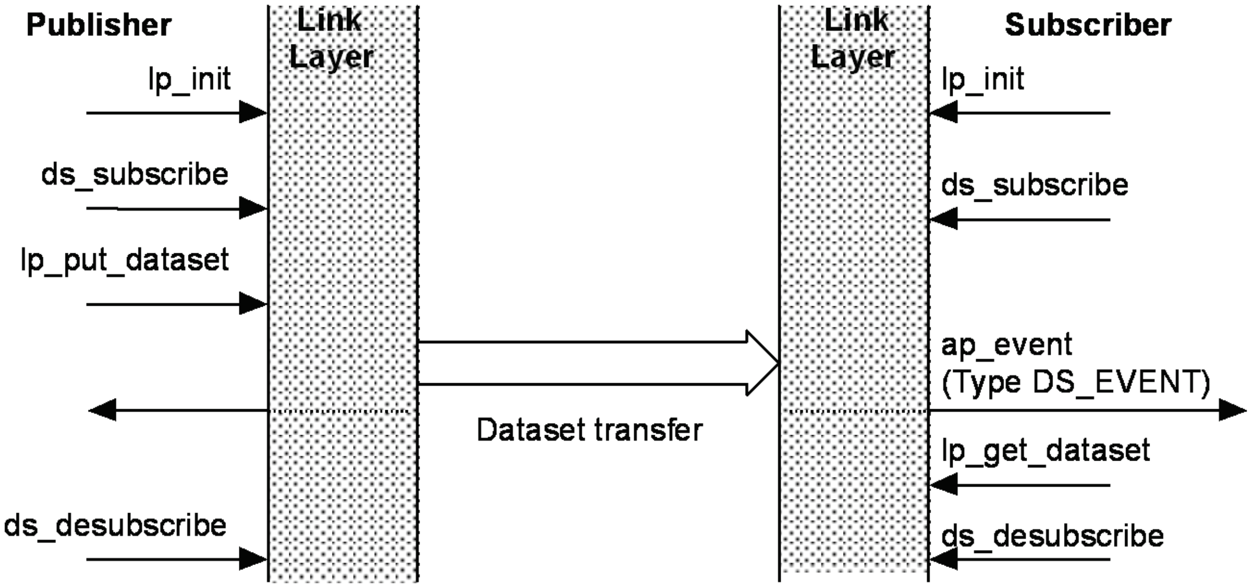


Figure 109 – LPI primitives exchange

Table 18 – LPI primitives

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

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

6.2.2.3.2 Type ‘LP_RESULT’

Definition	A procedure of the LPI which returns a value of type LP_RESULT shall encode it as follows:
Syntax	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;

6.2.2.3.3 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	LP_RESULT lp_init (ENUM8 ts_id void * p_descriptor);	
Input	ts_id	Traffic_Store_Id (0..15)
	p_descriptor	Implementation-dependent data structure.
Return		any LP_RESULT

6.2.2.3.4 Procedure ‘lp_put_dataset’

Definition	Copies a Dataset from the Application to a port in the Traffic_Store.	
Syntax	LP_RESULT lp_put_dataset (DS_NAME * dataset; void * p_value);	
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.	

6.2.2.3.5 Procedure ‘lp_get_dataset’

Definition	Copies a Dataset and its Freshness_Timer from a port to the Application.	
Syntax	`LP_RESULT lp_get_dataset ( DS_NAME * dataset; void * p_value; void * p_fresh );`	
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.

```
LP_RESULT          lp_get_dataset
                   (
    DS_NAME *      dataset;
    void *         p_value;
    void *         p_fresh
                   );
```

6.2.2.3.6 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	`LP_RESULT Ds_subscribe ( DS_NAME * dataset; DS_EVENT event_cnf; UNSIGNED16 instance );`	
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.	

```
LP_RESULT          Ds_subscribe
                   (
    DS_NAME *          dataset;
    DS_EVENT           event_cnf;
    UNSIGNED16         instance
                   );
```

6.2.2.3.7 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	`typedef void ( * DS_EVENT)`
	`(`
	`UNSIGNED16 instance`
	`);`	

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.	

6.2.2.3.8 Procedure 'ds_desubscribe'

Definition	Removes a Dataset from the subscription.	

Syntax	`LP_RESULT Ds_desubscribe`
	`(`
	`DS_NAME * dataset;`
	`);`	

Input	dataset	DS_Name of the Dataset to be removed from subscription
Return		Any LP_RESULT

6.2.3 Application interface for Process_Variables

6.2.3.1 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).

6.2.3.2 Process_Variables

6.2.3.2.1 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.

6.2.3.2.2 Freshness supervision

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

6.2.3.2.3 Synchronisation

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

6.2.3.2.4 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 ANTIVALENT2 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 110 shows a Process_Variable and its associated Check_Variable in the same Dataset.

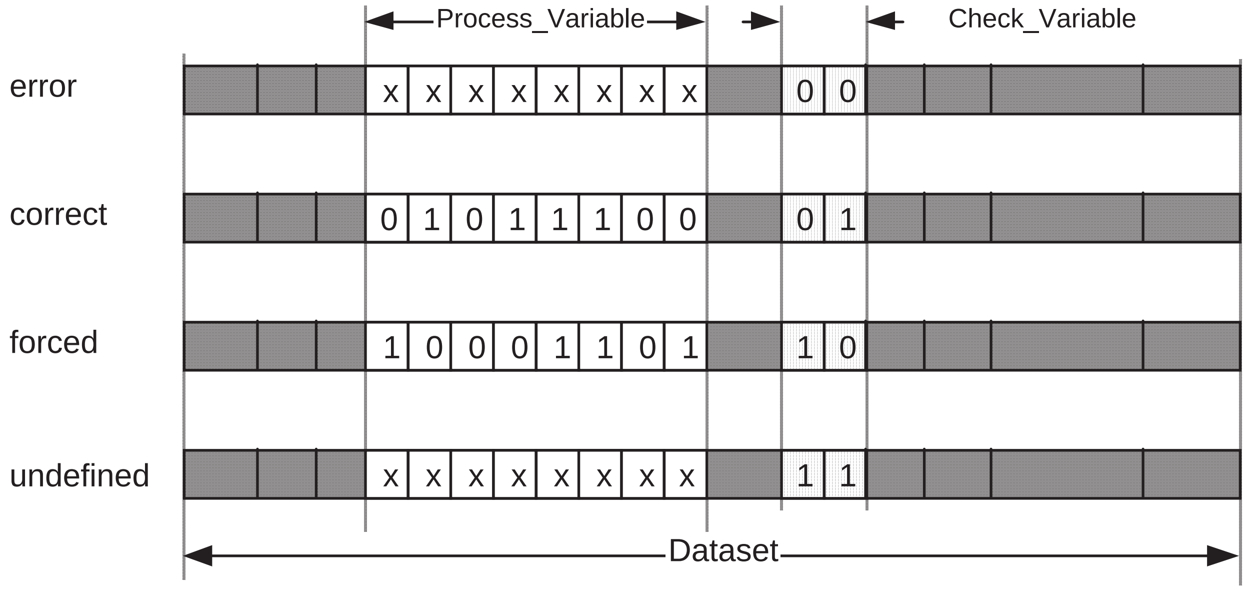


Figure 110 – Check_Variable

6.2.3.2.5 Process_Variable identifier (PV_Name)

6.2.3.2.5.1 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).

6.2.3.2.5.2 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:

a) Traffic_Store_Id;
b) Port_Address;
c) Var_Offset;
d) Var_Size;
e) Var_Type;
f) 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.

6.2.3.2.5.3 Traffic_Store Identifier

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

6.2.3.2.5.4 Port_Address

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

6.2.3.2.5.5 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.

6.2.3.2.5.6 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 6.4 and 'Var_Size' the size.

'Var_Size' and 'Var_Type' shall be encoded as shown in Table 19.

Table 19 – 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 × 16 = 1024 bits), Var_Size being '3F'H.

NOTE 4 Codes not mentioned in Table 19 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.

6.2.3.2.5.7 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.

6.2.3.3 Application_Variables_Interface primitives

6.2.3.3.1 General

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

a) individual access;

b) Set access;

c) Cluster access.

The following subclauses do not imply a particular implementation. Any interface which provides the same semantics is allowed.

6.2.3.3.2 Type ‘AP_RESULT’

Definition	A procedure of the AVI which returns a value shall encode it as follows:

Syntax	`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;`

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

6.2.3.3.3 Configuration Parameters

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

6.2.3.3.4 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’.

6.2.3.3.5 Individual access primitives

The Application Layer for Variables (AVI) shall provide for individual Variable access the following primitives, illustrated in Figure 111 and specified in the following subclauses:

a) ap_put_variable,

b) ap_get_variable,

c) ap_force_variable,

d) ap_unforce_variable,

e) ap_unforce_all.

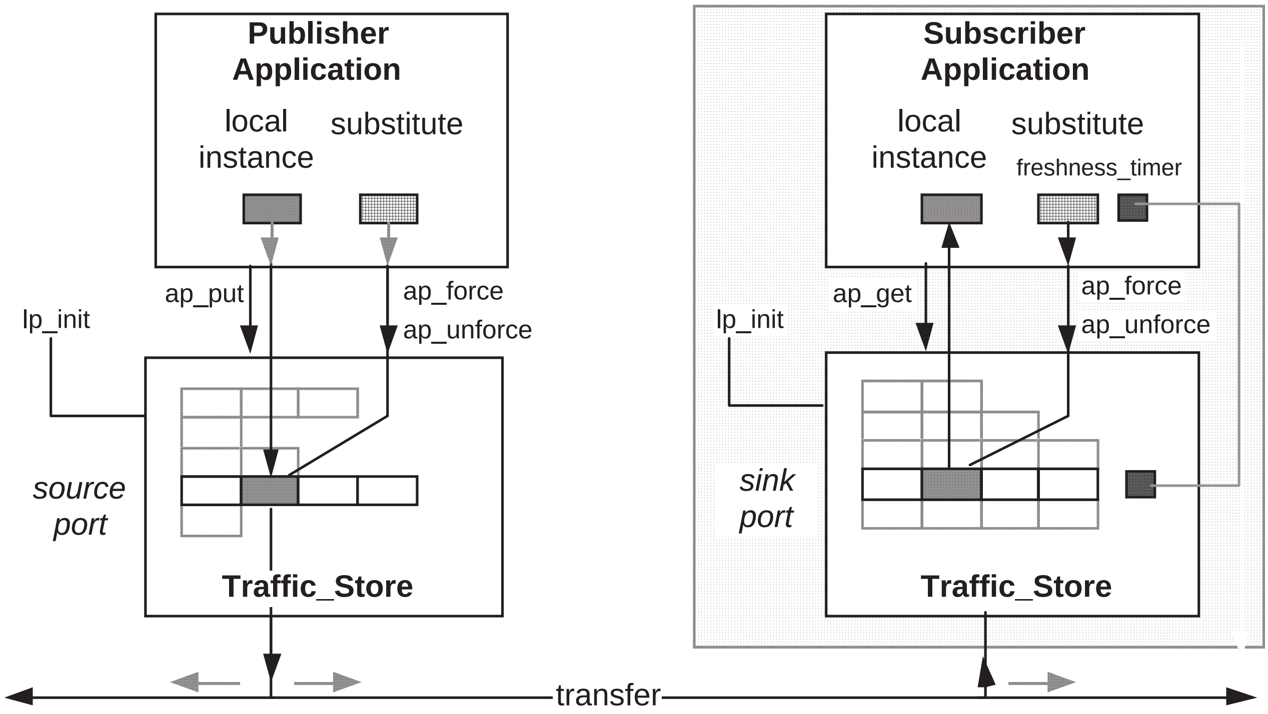


Figure 111 – Individual access

6.2.3.3.5.1 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

```
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 19   */
  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 19   */
  unsigned chk_octet_offset:7,   /* first octet has offset 0
    */
  unsigned chk_bit_number  :3,   /* counted from the right */
  } PV_NAME;
```

The PV_Name can be conveniently encoded as follows:

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="4">Traffic_store_id</td><td colspan="12">port_address</td></tr>
<tr><td colspan="6">var_size</td><td colspan="7">var_octet_offset</td><td colspan="3">var_bit_number</td></tr>
<tr><td colspan="6">var_type</td><td colspan="7">chk_octet_offset</td><td colspan="3">chk_bit_number</td></tr>
</table>

NOTE The definition of Var_Bit_Number was introduced to speed up individual access and especially avoid to add or substract 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:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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 × 8 = 248) bits;
- it is of type INTEGER8 (var_size = 0 × 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.

6.2.3.3.5.2 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	`AP_RESULT`
	`PV_NAME*`
	`void *`
	`void *`
		`ap_put_variable`
		`(`
		`ts_variable,`
		`p_value,`
		`p_check`
		`);`

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.	

6.2.3.3.5.3 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	`AP_RESULT`
	`PV_NAME*`
	`void *`
	`void *`
	`void *`
		`ap_get_variable`
		`(`
		`ts_variable,`
		`p_value,`
		`p_check,`
		`p_fresh`
		`);`

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.	

6.2.3.3.5.4 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	`AP_RESULT ap_force_variable (`
	`PV_NAME * ts_variable,`
	`void * p_value`
	`);`	

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.	

6.2.3.3.5.5 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	`AP_RESULT ap_unforce_variable (`
	`PV_NAME * ts_variable`
	`);`	

Input	ts_variable	PV_Name of the Process_Variable
Return		any AP_RESULT

6.2.3.3.5.6 Procedure ap_unforce_all

Definition	Terminates the substitution of all Variables of a Traffic_Store, does not modify the corresponding Check_Variables.	

Syntax	`AP_RESULT ap_unforce_all (`
	`ENUM8 ts_id`
	`);`	

Input	ts_id	Traffic_Store_Id (0..15)
Return		any AP_RESULT

6.2.3.3.6 Set access procedures

6.2.3.3.6.1 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 112 and specified in the following subclauses:

a) ap_put_set,

b) ap_get_set.

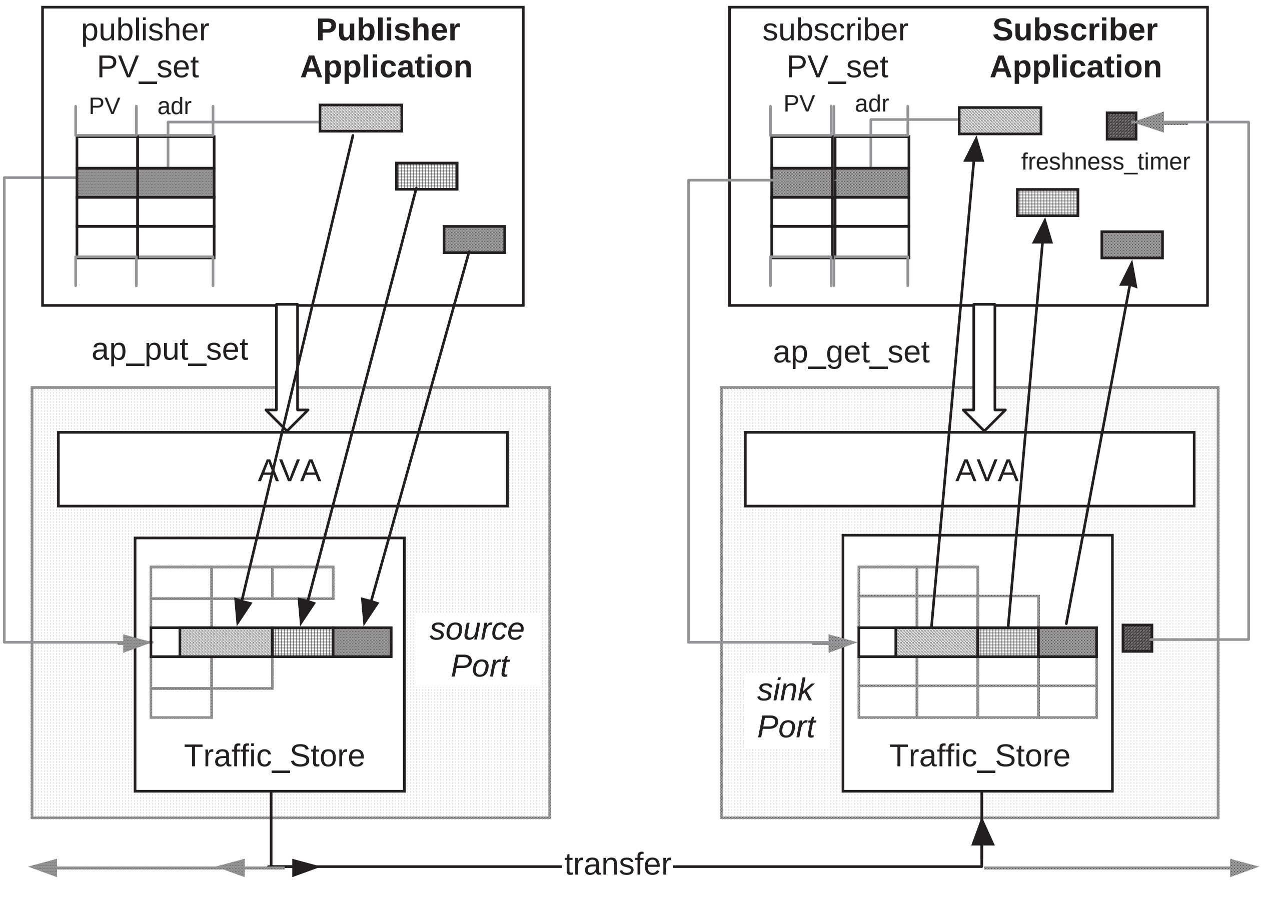


Figure 112 – Set access

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

6.2.3.3.6.2 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 PV_SET {`
	`struct PV_LIST* p_pv_list;`
	`UNSIGNED16 c_pv_list;`
	`UNSIGNED16 * p_freshtime;`
	`DS_NAME 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.

6.2.3.3.6.3 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	`AP_RESULT ap_put_set (PV_SET * pv_set );`	
Input	pv_set	pointer to a PV_List
Return		any AP_RESULT

6.2.3.3.6.4 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	`AP_RESULT ap_get_set (PV_SET * pv_set );`	
Input	pv_set	pointer to a PV_List
Return		any AP_RESULT

6.2.3.3.7 Cluster access procedures

6.2.3.3.7.1 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 113 and specified in the following subclauses:

- ap_put_cluster,
- ap_get_cluster.

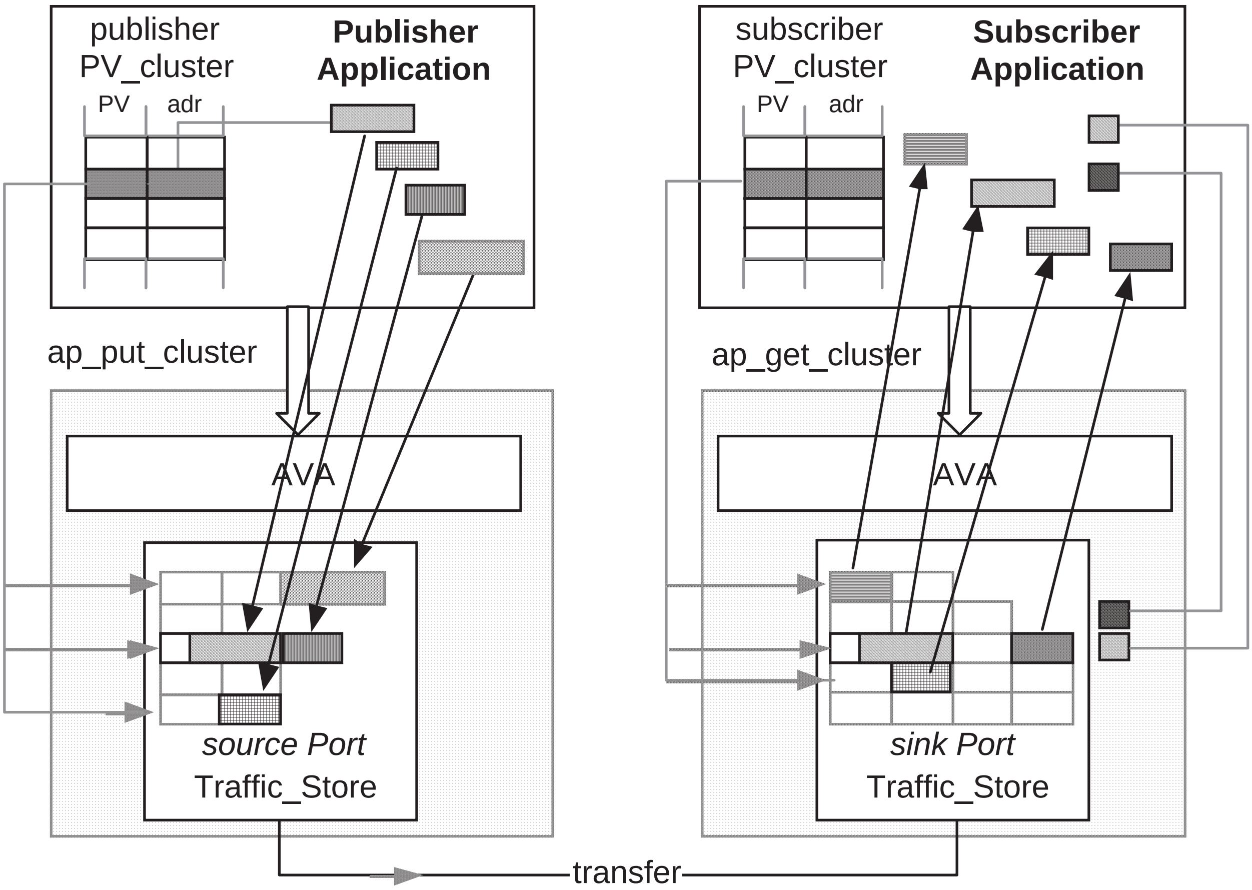


Figure 113 – Cluster access

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.

6.2.3.3.7.2 Type PV_CLUSTER

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

Definition	Type of a PV_Cluster	
Syntax	`typedef struct` `UNSIGNED8` `UNSIGNED8` `struct PV_SET *`	`PV_CLUSTER` `{` `ts_id;` `c_pv_set;` `p_pv_set [c_pv_set]` `}`
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.	

```
typedef struct     PV_CLUSTER
                   {
UNSIGNED8          ts_id;
UNSIGNED8          c_pv_set;
struct PV_SET *    p_pv_set [c_pv_set]
                   }
```

6.2.3.3.7.3 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	`AP_RESULT` `PV_CLUSTER*`	`ap_put_cluster` `(` `pv_cluster` `);`
Input	pv_cluster	pointer to a Publisher Cluster List
Return		any AP_RESULT

```
AP_RESULT          ap_put_cluster
                   (
    PV_CLUSTER*    pv_cluster
                   );
```

6.2.3.3.7.4 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	`AP_RESULT` `PV_CLUSTER*`	`ap_get_cluster` `(` `pv_cluster` `);`
Input	pv_cluster	pointer to a Subscriber Cluster List
Return		any AP_RESULT

```
AP_RESULT          ap_get_cluster
                   (
    PV_CLUSTER*    pv_cluster
                   );
```

6.3 Messages Services and Protocols

6.3.1 General

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

a) the lower, link layer interface, which specifies the services expected from the bus; and

b) the higher, application layer services are offered to the application.

6.3.2 Reference station

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

a) at least one bus connection, accessible through its LPI, and which implements a link layer Process;

b) a protocol machine, called the Messenger, implementing the network, transport, session, presentation and application layer;

c) 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 8 (Train_Network_Management).

6.3.2.1 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 114.

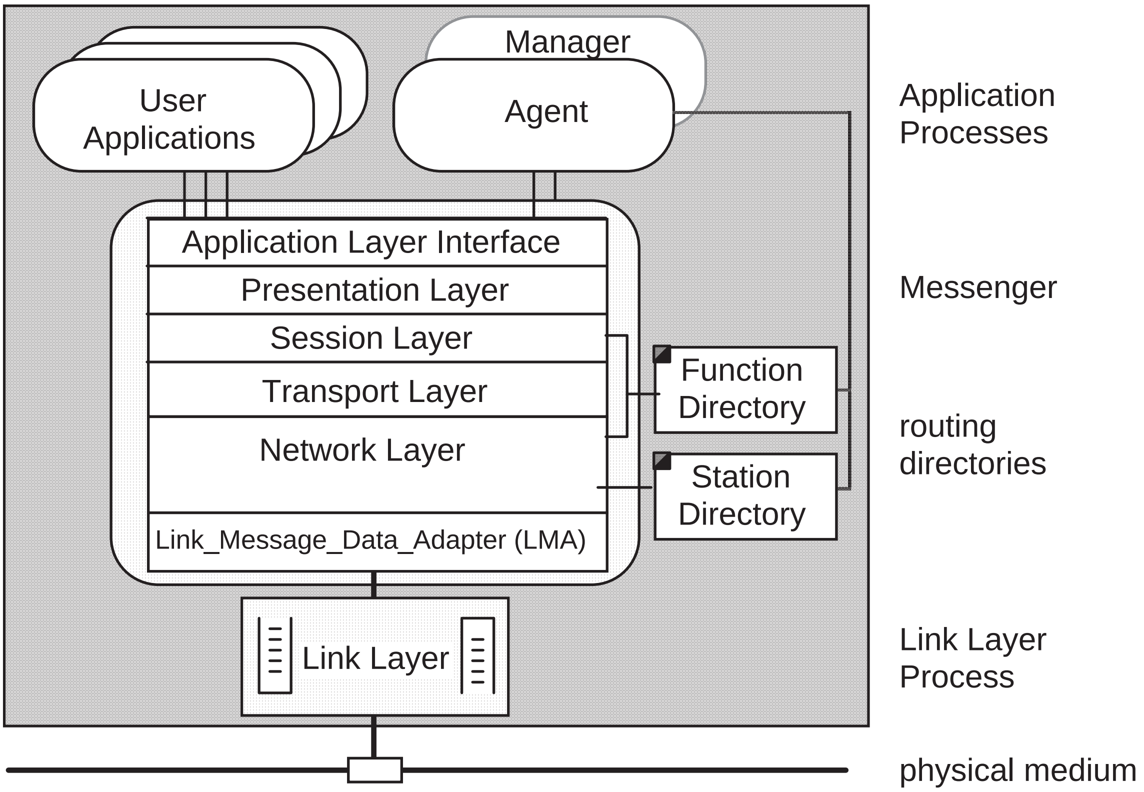


Figure 114 – Terminal station

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

6.3.2.2 Router station

Station attached to several busses, or Router station, shall have one link layer for each bus. Both busses share the same Real-Time Protocols.

EXAMPLE Figure 115 shows a Router station attached to the link layer of an MVB and of a WTB.

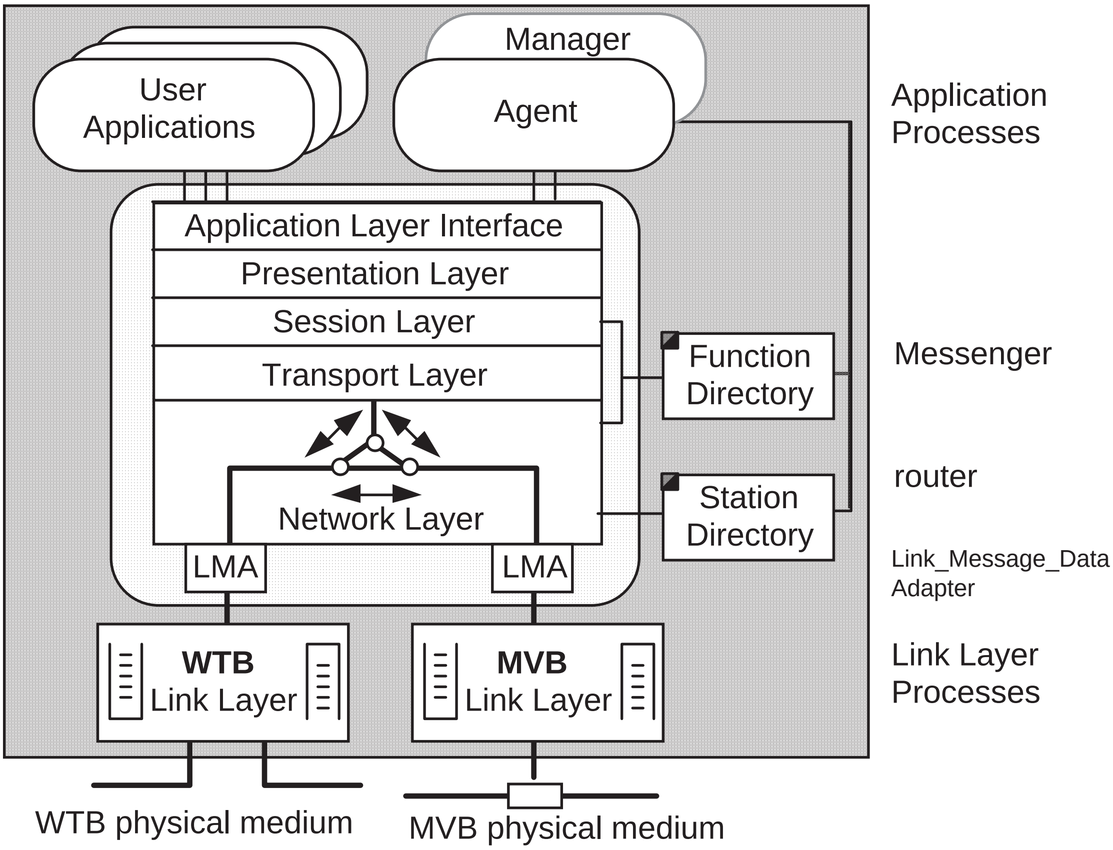


Figure 115 – Router station between WTB and MVB

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

6.3.2.3 Gateway station

Station attached to several busses, or Gateway station, shall have one link layer for each bus. The Consist network has another protocol than WTB RTP. The Gateway station has to adapt the Consist network protocols to WTB RTP.

EXAMPLE Figure 116 shows a Gateway station attached to the link layer of a consist network and of a WTB.

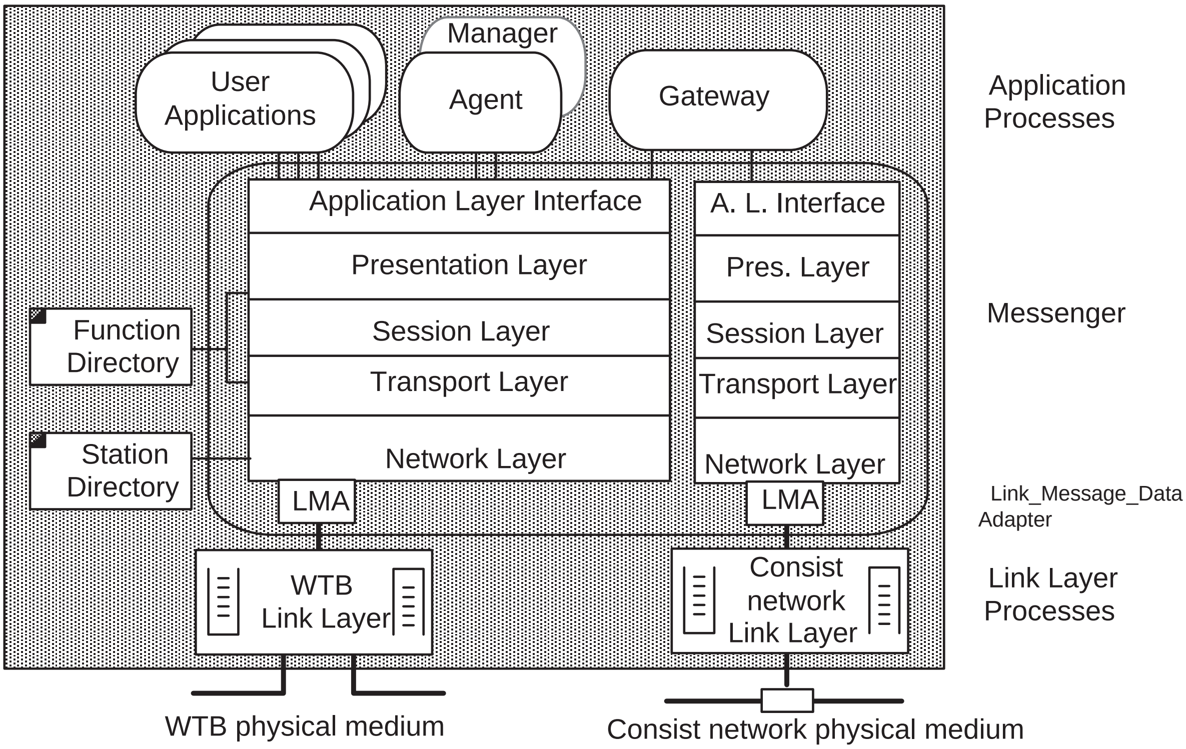


Figure 116 – Gateway station between WTB and Consist network

6.3.2.4 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.

6.3.2.5 Bus identifier

A station shall identify each link layer, i.e. each bus (consist network 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.

6.3.2.6 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.

6.3.3 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. The following subclauses do not imply a particular implementation. Any interface which provides the same semantics is allowed.

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

6.3.3.1 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.

6.3.3.2 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	`typedef void* MD_PACKET;`
Usage	The format of MD_PACKET is not prescribed.

EXAMPLE An example of MD_PACKET is shown in Figure 117.

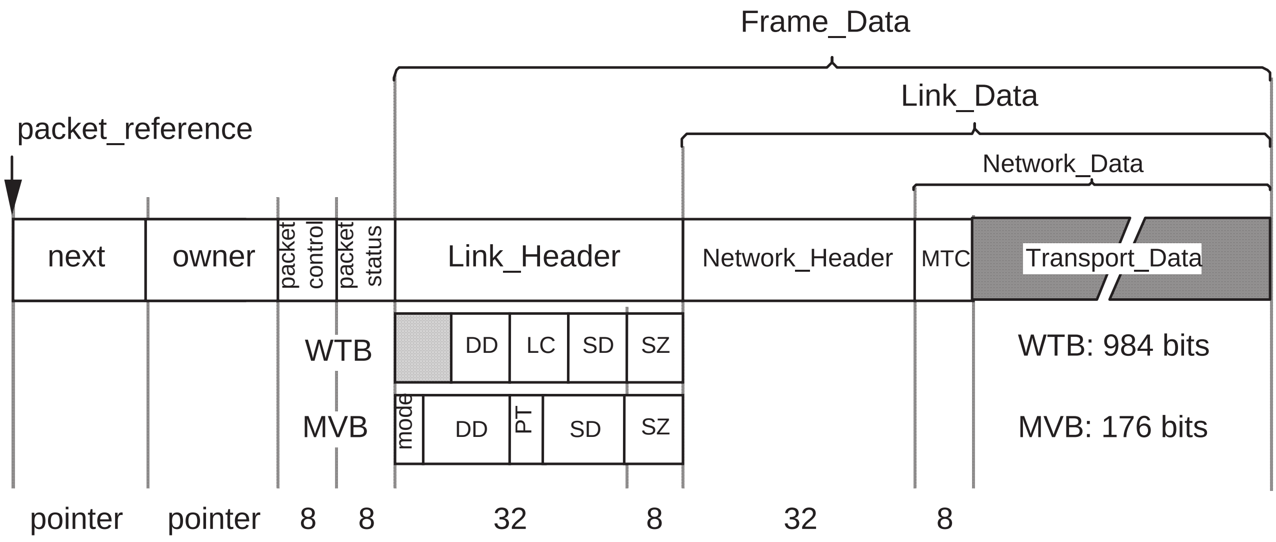


Figure 117 – Packet format

The example packet of Figure 117 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.

6.3.3.3 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	`typedef void ( * MD_GET_PACKET ) (`
	`    void * *         pool,`
	`    MD_PACKET * *      packet`
	`);`	

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.	

6.3.3.4 Procedure Type ‘MD_PUT_PACKET’

Definition	returns a single packet not used anymore to the pool specified by the owner field of the packet.	

Syntax	`typedef void ( * MD_PUT_PACKET ) (`
	`    MD_PACKET *        packet`
	`);`	

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.	

6.3.4 Message Link layer

6.3.4.1 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 118.

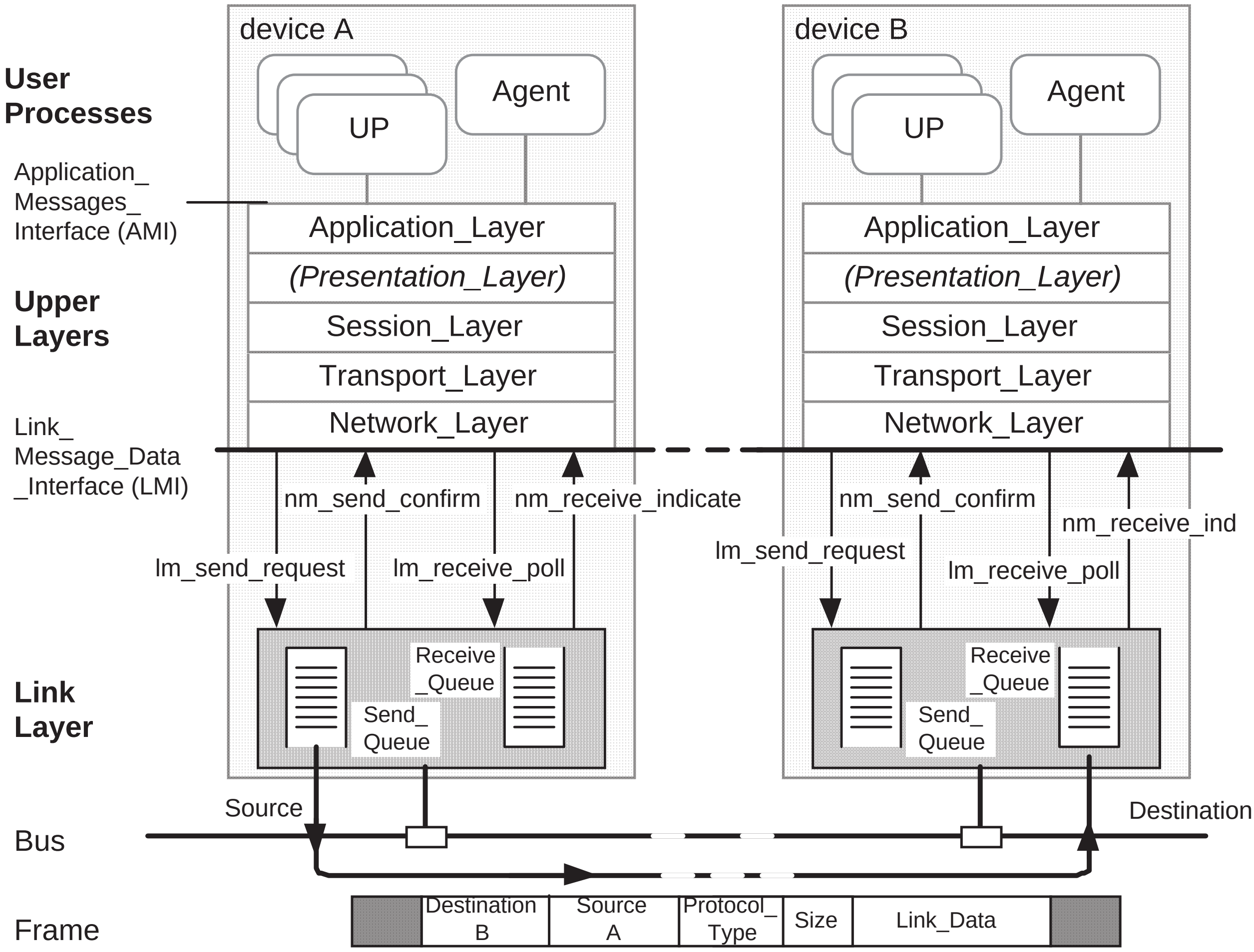


Figure 118 – Link layer data transmission

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.

6.3.4.2 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 119.

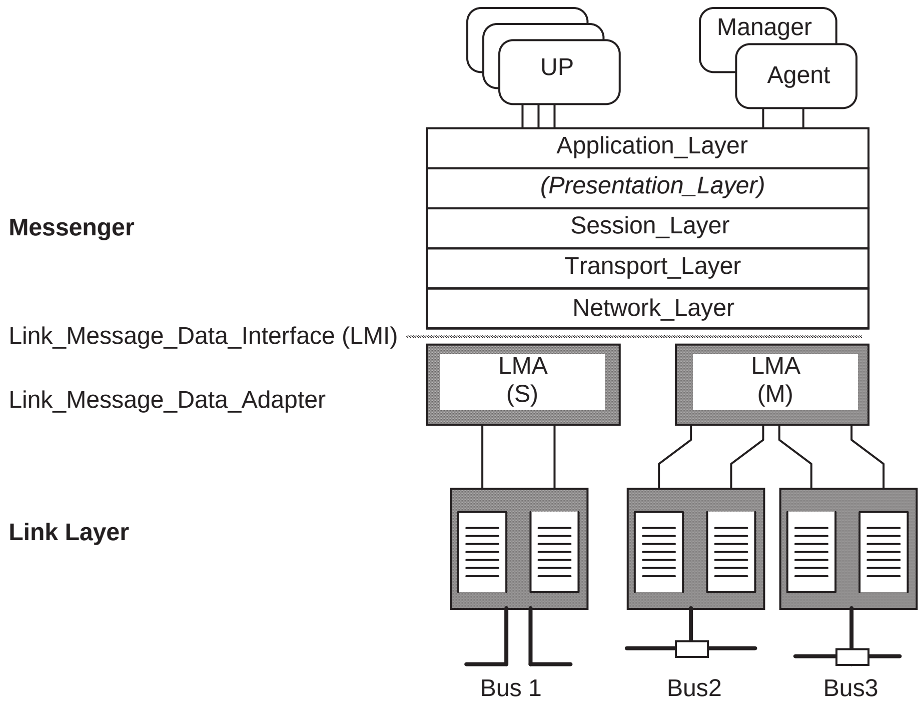


Figure 119 – Link_Message_Data_Interface (LMI)

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).

6.3.4.3 Link layer characteristics

6.3.4.3.1 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.

6.3.4.3.2 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_Queues and the Receive_Queues used for the Real-Time Protocols.

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

6.3.4.3.3 Priorities

Priorities are not distinguished at the link layer.

6.3.4.3.4 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.

6.3.4.3.5 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 an individual invalidation of the packet in the queue.

6.3.4.3.6 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.

6.3.4.4 Link layer of the consist network, example MVB

The consist network may be implemented by different busses, e.g. by the MVB (see IEC 61375-3-1), which serves as a reference.

The Device_Address of a consist network device shall not change during regular operation.

If the MVB is used as a consist network, 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 in Figure 120.

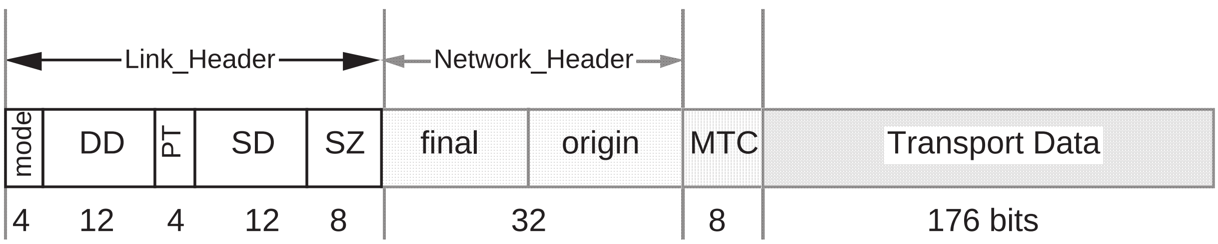


Figure 120 – Example of MVB Message_Data frame

6.3.4.5 Link layer of the train bus

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

6.3.4.5.1 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. 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 consist. The mapping of consist 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 = ‘11111111’B.
– The Protocol_Type of the link layer for the Real-Time Protocols is identified by Link_Control = ‘x0xxx111’B (Message_Data).

EXAMPLE Figure 121 shows the frame format of the WTB.

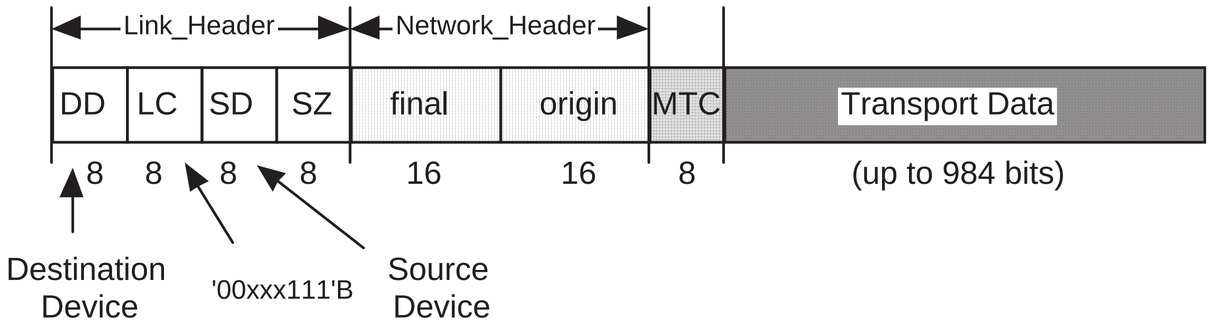


Figure 121 – Example of WTB Message_Data frame

6.3.4.6 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.

The following subclauses do not imply a particular implementation. Any interface which provides the same semantics is allowed.

6.3.4.6.1 LMI primitives

The Link_Message_Data_Interface shall provide the primitives illustrated in Figure 122, which are listed in Table 20 and specified in the following subclauses.

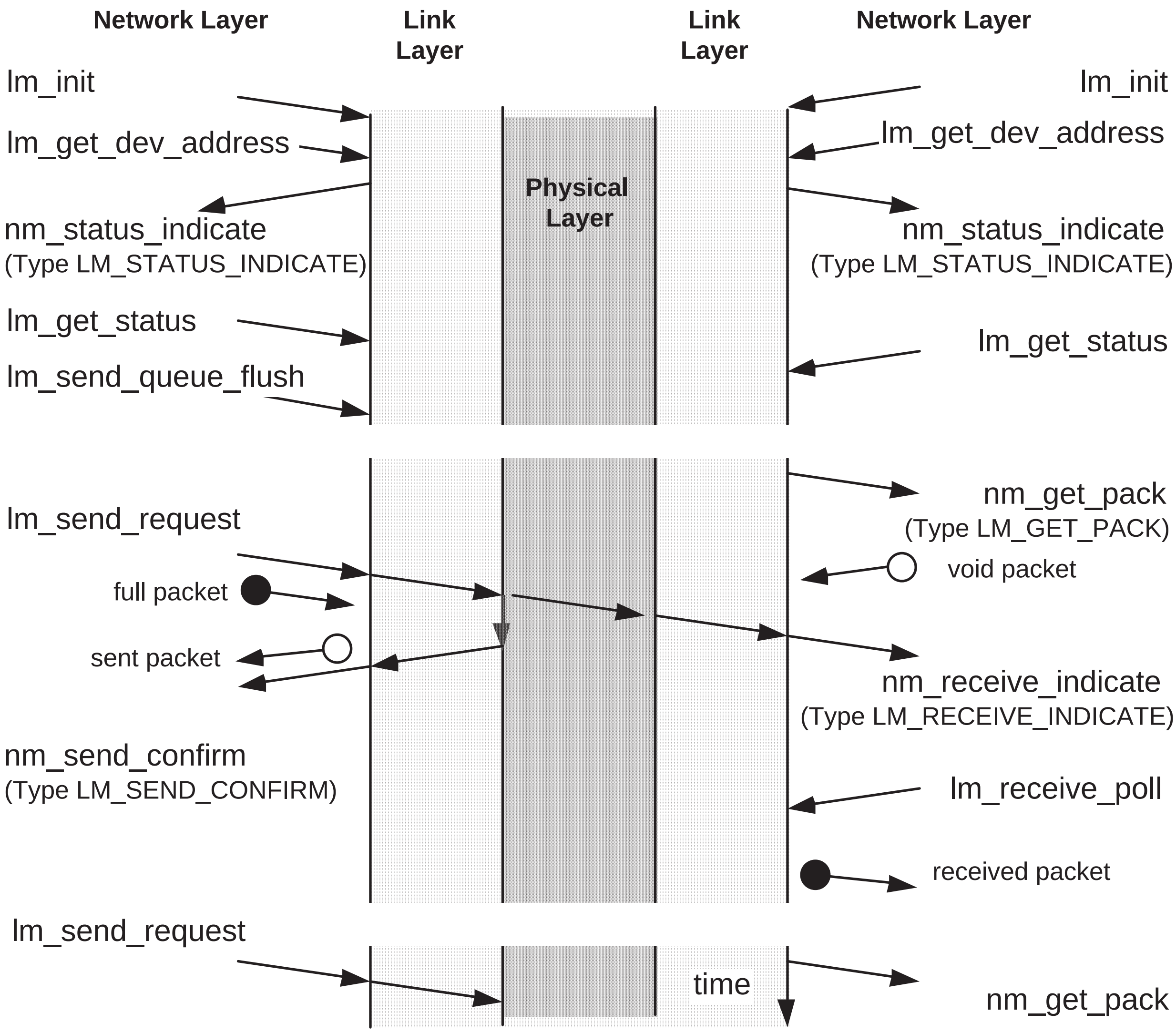


Figure 122 – LMI primitives

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 20 – 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

6.3.4.6.2 Type 'MD_RESULT'

Definition	A procedure of the LMI which returns a value shall encode it as follows:

Syntax	`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;`

6.3.4.6.3 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

6.3.4.6.4 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	`typedef void ( * LM_SEND_CONFIRM )`
	`(`
	`MD_PACKET * packet`
	`);`	

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’.	

6.3.4.6.5 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	`typedef void ( * LM_GET_PACK )`
	`(`
	`void * * owner,`
	`MD_PACKET * * packet`
	`);`	

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.	

6.3.4.6.6 Procedure type ‘LM_RECEIVE_INDICATE’

Definition	When a packet is received, the link layer shall call a procedure of this type.	

Syntax	`typedef void ( * LM_RECEIVE_INDICATE )`
	`(`
	`ENUM8 bus_id`
	`);`	

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.	

6.3.4.6.7 Procedure ‘lm_receive_poll’

Definition	Reads one received packet out of the Receive_Queue and pass the packet reference to the network layer.	

Syntax	`MD_RESULT lm_receive_poll`
	`(`
	`ENUM8 bus_id,`
	`unsigned * source,`
	`unsigned * destination,`
	`MD_PACKET * * packet`
	`);`	

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’.	

6.3.4.6.8 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	`typedef void ( * LM_STATUS_INDICATE )`
	`(`
	`ENUM8 bus_id,`
	`MD_RESULT status`
	`);`	

Input	bus_id	Bus_Id (0..15)
	status	MD_OK (regular operation) 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.	

6.3.4.6.9 Procedure 'lm_get_status'

Definition	Retrieves detailed information on the link layer.	
Syntax	`MD_RESULT lm_get_status ( ENUM8 bus_id, BITSET8 selector, BITSET8 reset, BITSET8 * status );`	
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 = 1 set whenever a correct frame is received from the bus. MD_SEND_ACTIVE = 2 set when a frame has been transmitted. MD_RECEIVE_OVERFLOW = 4 set 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 consist network link layer.	

```
MD_RESULT          lm_get_status
                   (
    ENUM8          bus_id,
    BITSET8        selector,
    BITSET8        reset,
    BITSET8 *      status
                   );
```

6.3.4.6.10 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	`MD_RESULT lm_send_queue_flush ( ENUM8 bus_id /* optional */ );`	
Input	bus_id	Bus_Id (0..15)
Return		any MD_RESULT

```
MD_RESULT          lm_send_queue_flush
                   (
    ENUM8          bus_id  /* optional */
                   );
```

6.3.4.6.11 Procedure ‘lm_init’

Definition	Initialises the link layer, flushes the Send_Queue and subscribes the indication procedures from the network layer.	

Syntax	`MD_RESULT lm_init (`
	`ENUM8 bus_id,`
	`LM_RECEIVE_INDICATE nm_receive_indicate,`
	`LM_GET_PACK nm_get_pack,`
	`void * * owner,`
	`LM_SEND_CONFIRM nm_send_confirm,`
	`LM_STATUS_INDICATE nm_status_indicate`
	`);`	

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 uses 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

6.3.4.6.12 Procedure ‘lm_get_dev_address’

Definition	reads the own Device_Address corresponding to the specified link layer.	

Syntax	`MD_RESULT lm_get_dev_address (`
	`ENUM8 bus_id,`
	`UNSIGNED * device_address`
	`);`	

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.	

6.3.5 Message Network Layer

6.3.5.1 Purpose

The network layer relays packets, as illustrated in Figure 123:

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

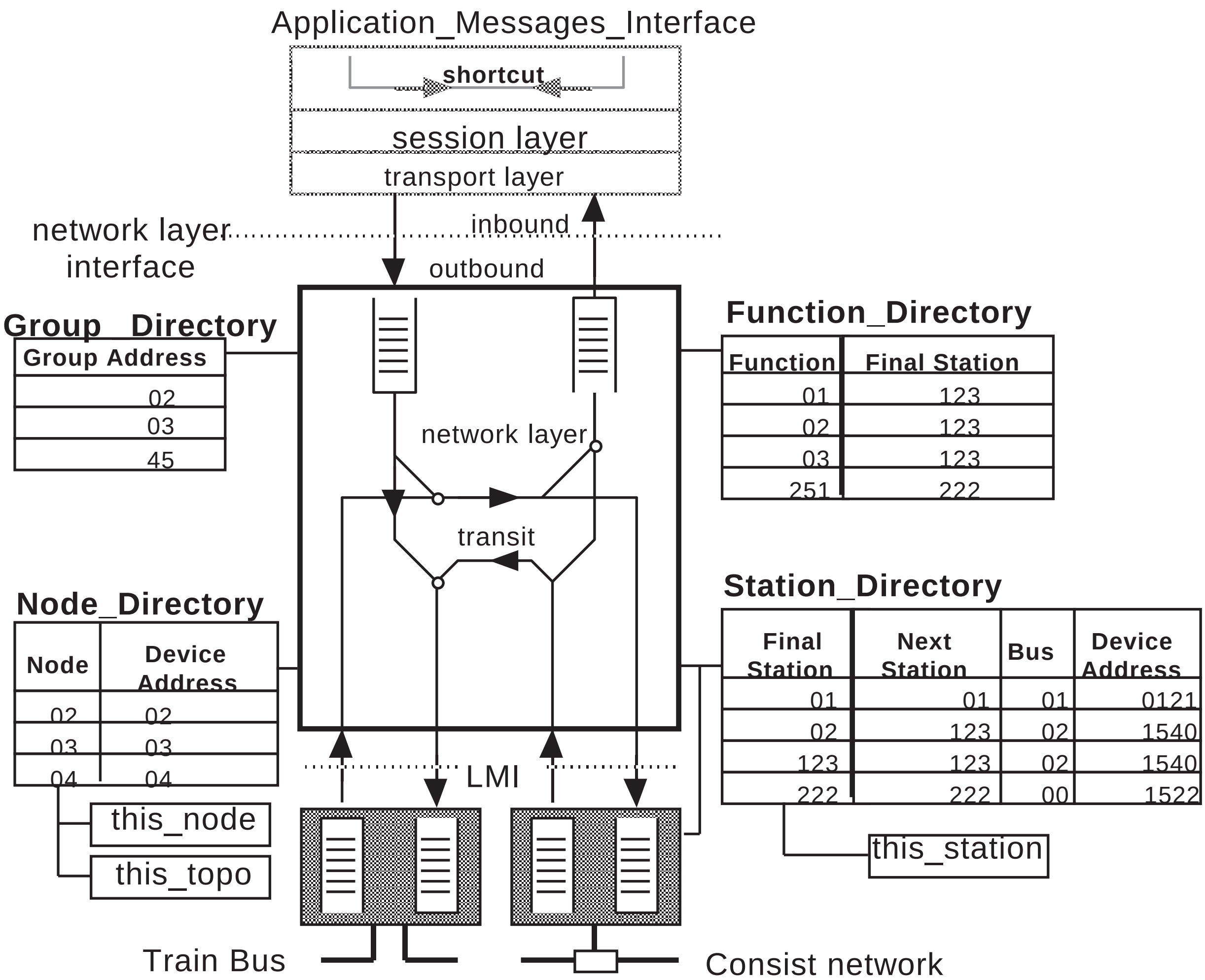


Figure 123 – Network layer on a Node

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:

a) the station directory,

b) the function directory,

c) the group directory, and

d) the node directory.

The network layer is connectionless.

6.3.5.2 Directories

6.3.5.2.1 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).

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 consist networks or between a consist network and a sensor bus.

NOTE 3 The access primitives of the station directory are defined in the Application Layer Interface (ALI).

6.3.5.2.2 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 (ALI).

6.3.5.2.3 Group directory

The Network Layer of a station participating in multicast communication shall indicate to which

Membership	list of Groups

Group this station pertains through a group directory, structured as follows:

NOTE The access primitives of the group directory are defined in the Application Layer Interface (ALI).

6.3.5.2.4 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 (ALI).

6.3.5.3 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.

6.3.5.4 Network_Address

6.3.5.4.1 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 124.

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

Final Network_Address

7	6	5	4	3	2	1	0
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 124 – Encoding of the Network_Address

6.3.5.4.2 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.

6.3.5.4.3 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.

6.3.5.4.4 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.

6.3.5.4.5 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.

6.3.5.4.6 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.

6.3.5.4.7 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).

6.3.5.5 Network Header encoding

6.3.5.5.1 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 125.

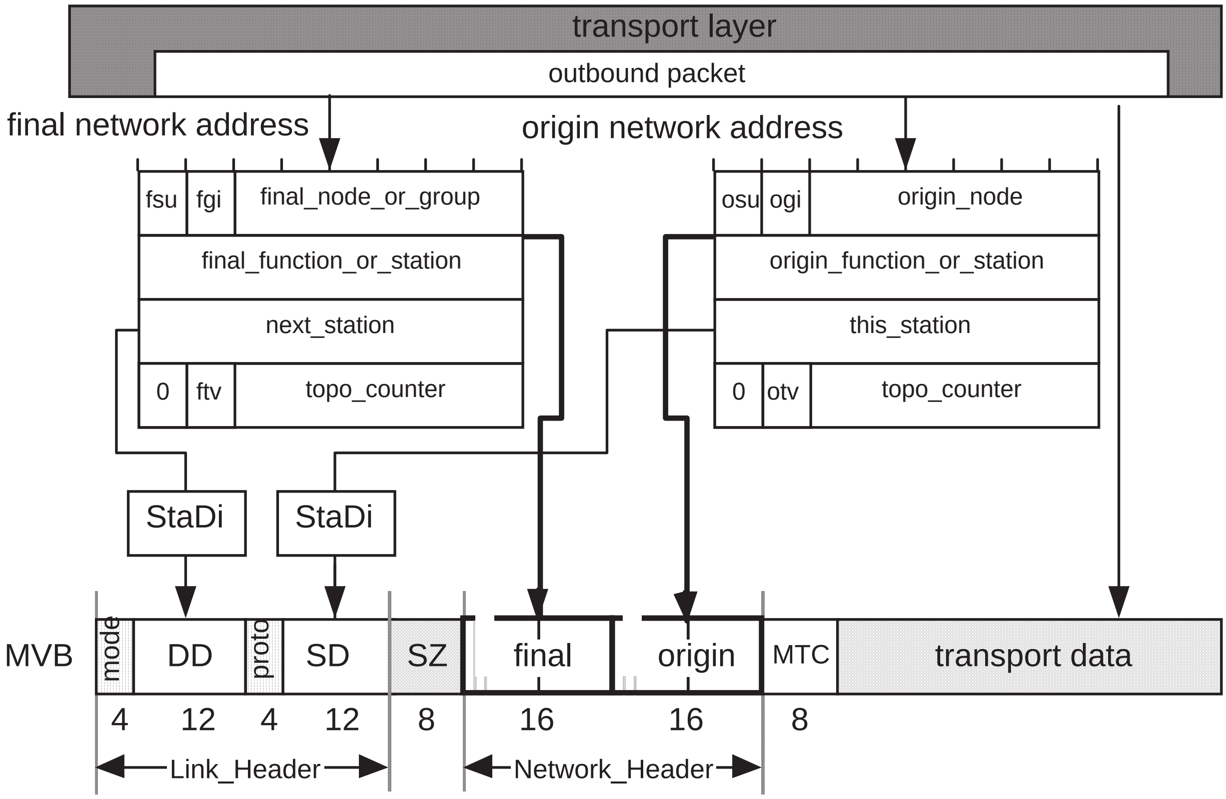


Figure 125 – Building of the addresses in an outbound packet

6.3.5.5.2 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 126 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
    }
  }
```

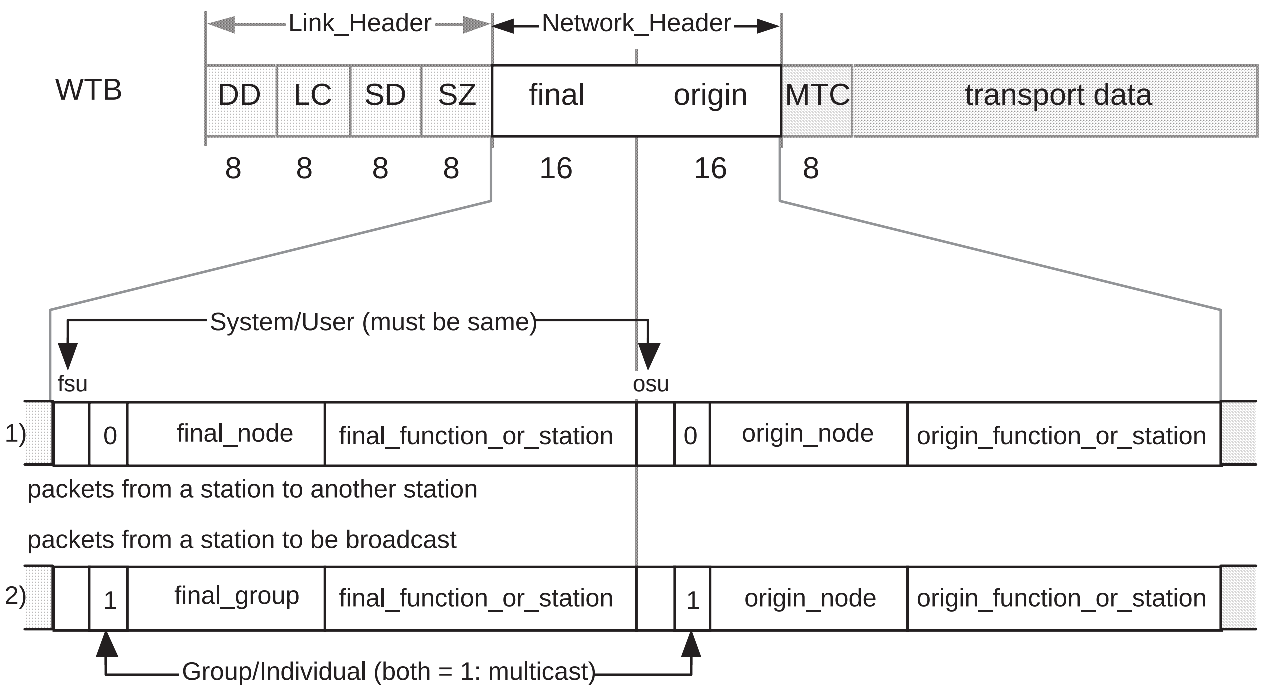


Figure 126 – Network address encoding on the train bus

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’.

6.3.5.6 Routing in the network layer

6.3.5.6.1 Situations

The nine situations listed in Table 21 define the behaviour of a full-fledged network layer connected to

- a transport layer;
- one or more consist network link layers;
- a train bus link layer.

Table 21 – Routing situations

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

A terminal station attached to the consist network 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 consist networks encounters only situations 1, 2, 4 and 5.

6.3.5.6.2 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.

6.3.5.6.3 Topography consistency

To safeguard against configuration changes of the train bus during a message transfer, packets sent to or from a Node over the consist network 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 consist network 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;
- 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 6.3.5.6.4.

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 6.3.5.6.4.

6.3.5.6.4 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 consist network and addressed to another consist 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.

6.3.5.6.5 Routing algorithms

A Router shall forward a packet as specified:

- for outbound packets in Table 22,
- for inbound and transit packets from a consist network in Table 23, and
- for inbound and transit packets coming from a train bus in Table 24.

Table 22 – 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 (fundi(function_id) = this_station)`

(2)	Consist network 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 23 – Routing of packets coming from a consist network

From	To	Condition

Consist network	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)`

4	transport layer	All cases not covered by the other conditions. Packet is 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.

5	another consist network	`( (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 24 – Routing of packets coming from the train bus

From	To	Condition
Train bus		The main difference with packets coming from a consist network 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	consist network	`((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.

6.3.5.7 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.

6.3.6 Message transport layer

6.3.6.1 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.

6.3.6.2 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 Application 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.

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.

6.3.6.3 Packet exchange

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

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

EXAMPLE Figure 127 shows a simple packet exchange with a window size of 1.

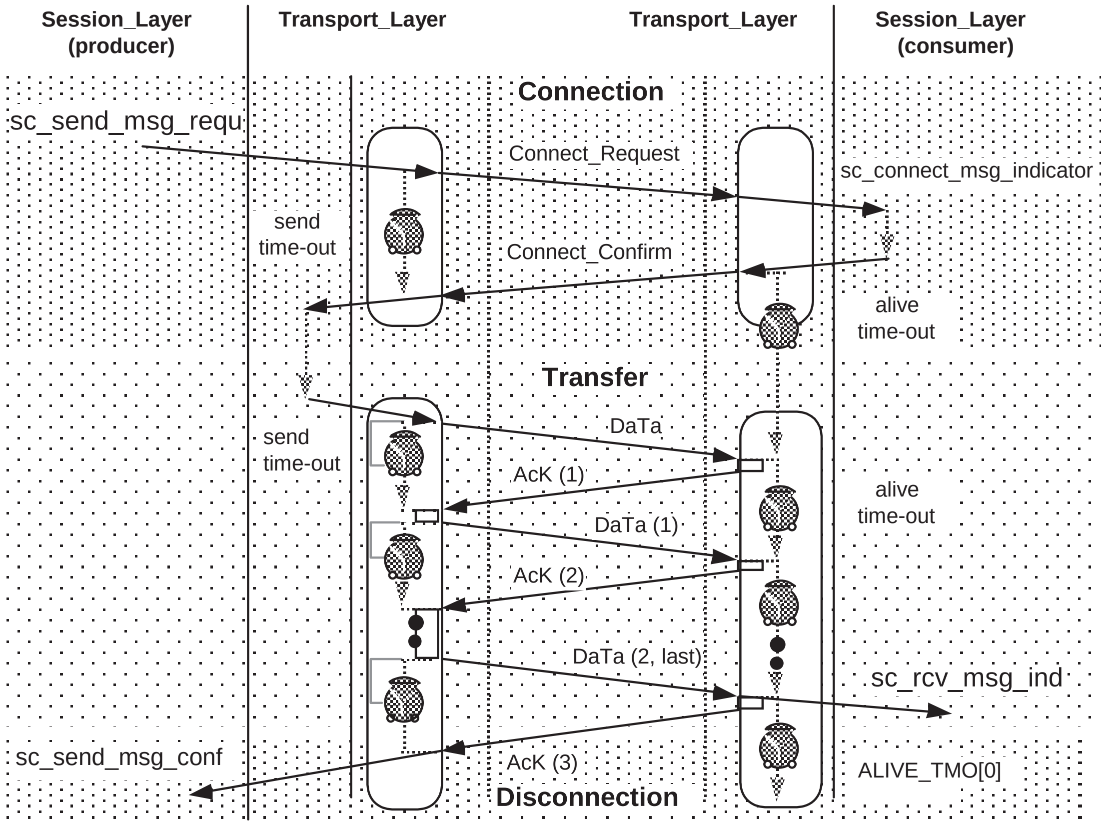


Figure 127 – Transport packet exchange

6.3.6.3.1 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.

6.3.6.3.2 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.

6.3.6.3.3 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.

6.3.6.4 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.

The Connection_Reference shall be used for call-reply-pairing by the caller application.

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.

6.3.6.5 Transport layer packets

6.3.6.5.1 Packet types

The transport layer shall operate with the seven packet types illustrated in Figure 128 and specified in the following subclauses.

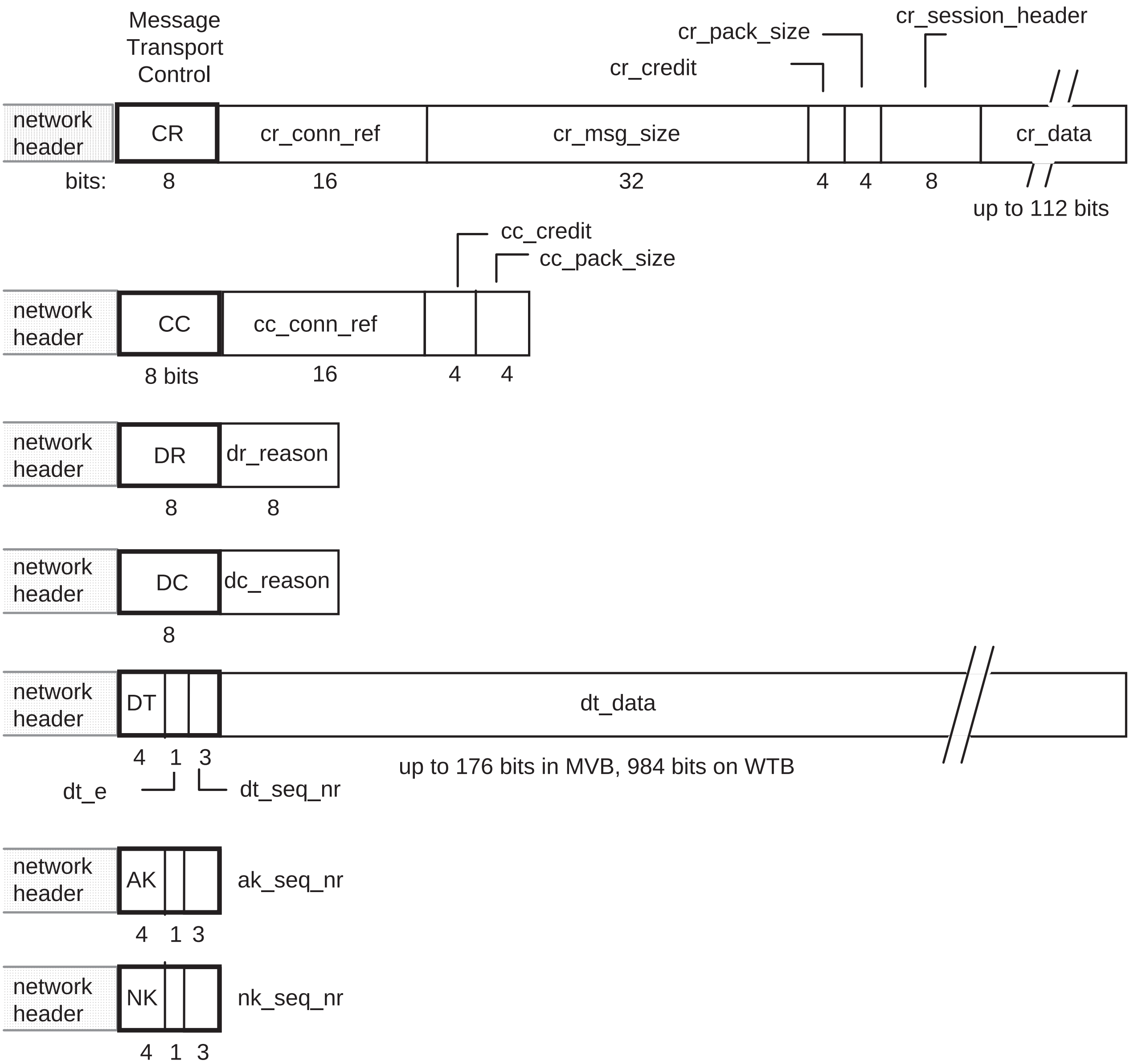


Figure 128 – Packet formats (transport layer body)

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).

6.3.6.5.2 Message Transport Control encoding

The Message Transport Control field shall be encoded as specified in Table 25.

Table 25 – Message Transport Control encoding

Packet type	Packet	Short	Encoding							
	Connect_Request	CR	cl	0	0	0	0	0	0	0
Control	Connect_Confirm	CC	cl	1	0	0	0	0	0	1
(single-cast)	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) datagram.

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.

6.3.6.5.3 Encoding of Am_Result

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

```
Am_Result::=           ENUM8
  {
  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)          -- 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
  AM_MAX_ERR           (31)         -- highest system code.
  }                                 -- user codes are higher than 31
```

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.

6.3.6.5.4 Packet encoding (see Table 26 to Table 36)

The Packets shall be formatted as specified in the following:

```
Message_Packet::= RECORD
  {
  call_not_reply       ENUM1               -- first bit distinguishes
                                           call from reply
    {
    REPLY_MSG  (0),                        -- reply message
    CALL_MSG   (1)                         -- call message
  consumer_not_producer                    ENUM1   -- second bit
                                           distinguishes consumer from
                                           producer
    {
```

```
    CONSUMER (0),                      -- coming from consumer
    PRODUCER (1)                       -- coming from producer
    },
  packet_kind         ENUM2            -- 3rd and 4th bits
                                          distinguish packet_kind
    {
    CONTROL (0)                        -- control message
    DATA    (1)                        -- data transfer
    ACK     (2)                        -- positive acknowledgement
    NAK     (3)                        -- negative acknowledgement
  ONE_OF [packet_type]
    {
    [CONTROL]         Control_Packet   -- control
    [DATA]            Data_Packet,     -- data transfer
    [ACK]             Ack_Packet,      -- positive acknowledgement
    [NAK]             Nak_Packet       -- negative acknowledgement
    }
  }

Control_Packet::= RECORD
  {
  command::==         ENUM4            -- last four bits of MTC octet
                                          indicate command
    {
    CR  (0),                           -- connect request
    CC  (1),                           -- connect confirm
    DR  (2),                           -- disconnect request
    DC  (3)                            -- disconnect confirm
    BC1 (8)                            -- broadcast request first
                                          trial  (optional)
    BC2 (9)                            -- broadcast request second
                                          trial (optional)
    BC3 (10)                           -- broadcast request third
                                          trial  (optional)
    BS  (12)                           -- broadcast stop  (optional)
    BR  (13)                           -- broadcast repeat (optional)
    BD  (14)                           -- broadcast data   (optional)
    },
  ONE_OF [command]                     -- depending on MTC
    {
    [CR]              Connect_Request,
    [CC]              Connect_Confirm,
    [DR]              Disconnect_Request,
    [DC]              Disconnect_Confirm,
    [BC1]             Broadcast_Connect,    -- see 6.3.7.5
    [BC2]             Broadcast_Connect,    -- see 6.3.7.5
    [BC3]             Broadcast_Connect,    -- see 6.3.7.5
    [BR]              Broadcast_Repeat,     -- see 6.3.7.5
    [BS]              Broadcast_Stop,       -- see 6.3.7.5
    [BD]              Broadcast_Data        -- see 6.3.7.5
    }
  }
```

Table 26 – Connect_Request

7	6	5	4	3	2	1	0
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 27 – Connect_Confirm

7	6	5	4	3	2	1	0
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 28 – Disconnect_Request

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

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

Table 29 – Disconnect_Confirm

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

```
Disconnect_Confirm::= RECORD
  {
  dc_reason           Am_Result           -- reason is AM_OK if
                                             disconnection successful
  }
```

Table 30 – Data_Packet

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

```
Data_Packet::= RECORD                     -- begins with last four bits
                                             of MTC octet
  {
  dt_e                BOOLEAN1            -- 1 = end of message (not of
                                             session)
  dt_seq_nr           UNSIGNED3           -- sequence number of this
                                             packet as seen by Producer.
  dt_data             ARRAY [transport_data_size] OF WORD8
  }                                       -- except possibly in the last
                                             of the message.
```

Table 31 – Ack_Packet

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

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

Table 32 – Nak_Packet

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

```
Nak_Packet::= RECORD
  {                                           -- last four bits of MTC octet
  nk_eot                BOOLEAN1              -- 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 33 – Broadcast_Connect (BC1, BC2, BC3)

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

```
Broadcast_Connect::= RECORD                   -- specified here for 6.3.7.5
  {                                           -- BC1, BC2 and BC3 are
                                                 distinguished by bc_rept
  bc_run_nr             UNSIGNED16            -- Connection_Reference
                                                 identifying this message
  bc_msg_size           UNSIGNED16            -- Total message size in
                                                 octets
  bc_data               ARRAY [18] OF WORD8     -- up to 18 octets of user
                                                 data
  }
```

Table 34 – Broadcast_Data

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

```
Broadcast_Data::= RECORD                         -- specified here for 6.3.7.5
  {
  bd_run_nr            UNSIGNED16                -- Connection_Reference
                                                    identifying this message
  bd_offset            UNSIGNED16                -- offset of bd_data in octets
                                                    within message
  bd_data              ARRAY [18] OF WORD8          -- up to 18 octets of user
                                                    data
  }
```

Table 35 – Broadcast_Repeat

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

```
Broadcast_Repeat::= RECORD                       -- specified here for 6.3.7.5
  {
  br_run_nr            UNSIGNED16                -- Connection_Reference
                                                    identifying this message
  br_offset            UNSIGNED16                -- offset in octets of missed
                                                    bd_data
  }
```

Table 36 – Broadcast_Stop (BSC, BSO)

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

```
Broadcast_Stop::= RECORD                    -- specified here for 6.3.7.5
  {
  bs_run_nr           UNSIGNED16            -- Connection_Reference
                                               identifying this message
  }
```

6.3.6.6 State transition diagrams

6.3.6.6.1 States

The Message Transport Protocol shall implement for each Producer or Consumer an MTP state machine with the states listed in Table 37.

Table 37 – 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 129 and specified in the following subclauses.

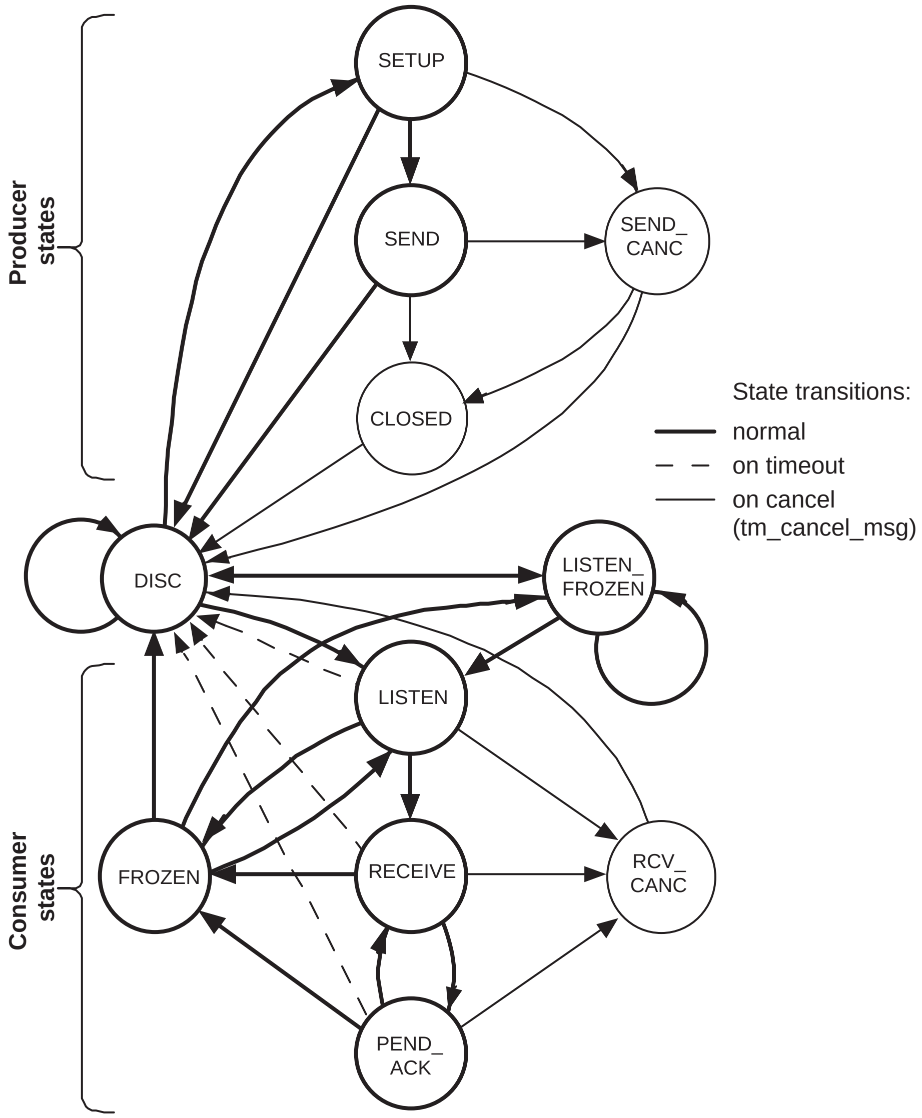


Figure 129 – State transition diagram of the MTP

6.3.6.6.2 Incoming events

The incoming events listed in Table 38, generated by the network, the user or time-outs, may cause transitions from one MTP state to another.

Table 38 – 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)

6.3.6.6.3 Outgoing events

The outgoing events listed in Table 39 generated by the MTP state machine, may cause transitions from one MTP state to another in another MTP machine.

Table 39 – 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

6.3.6.6.4 Control parameters in the packets

The exchanged packets shall contain the control parameters listed in Table 40.

Table 40 – MTP control parameters

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

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.

6.3.6.6.5 Auxiliary variables

The state transition diagram relies the auxiliary variables listed in Table 41 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 41 – MTP auxiliary variables

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

6.3.6.6.6 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.

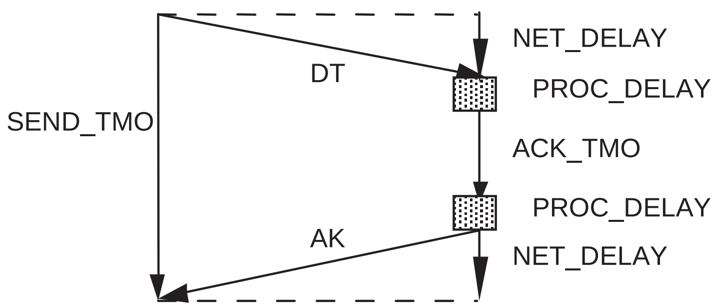
The Consumer shall acknowledge packets immediately when the credit is exhausted or when the message is completely received.

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 130.



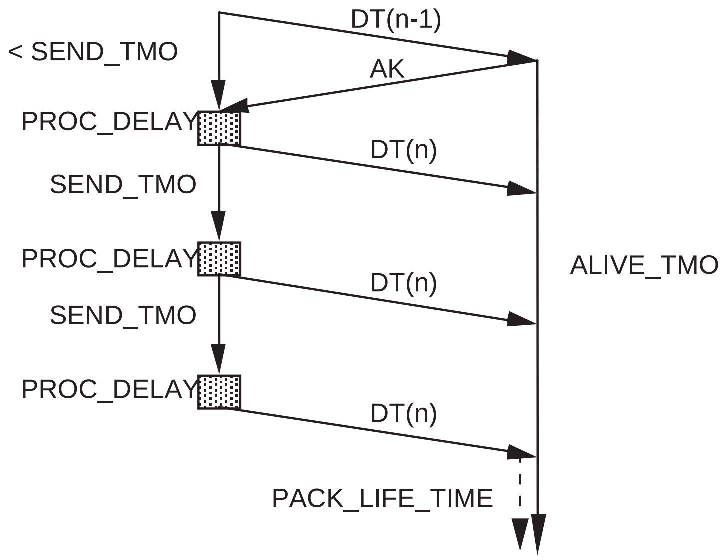
SEND_TMO = 2 × (NET_DELAY + PROC_DELAY) +

Figure 130 – 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 131.



ALIVE_TMO = MAX_REP_CNT × (SEND_TMO + PROC_DELAY) +

Figure 131 – Time-out ALIVE_TMO

The default values for a worst-case application are summarised in Table 42.

Table 42 – 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
consist network to consist network 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.

6.3.6.6.7 Implicit actions

Action descriptions in the following state transitions may contain the implicit actions referenced in Table 43.

Table 43 – 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.

6.3.6.6.8 Compound actions

The actions listed in Table 44 are executed in several state transitions.

Table 44 – 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;	-

close_send (error)	IF (NOT cancelled) THEN
	tm_send_cnf (error);
	END;
	reset_tmo;
		DISC

close_rcv (error)	IF (NOT cancelled) THEN
	tm_receive_ind (error);
	END;
	IF (error <> AM_OK) THEN
	reset_tmo;
	ELSE
	restart_tmo (ALIVE_TMO);
	END;
		DISC

update (eot)	eot:= message complete with next packet;	

send_data_or_cancel	IF cancelled THEN
	send_DR (AM_REM_CANC_ERR);
	rep_cnt:= 0;
	restart_tmo (SEND_TMO);
	ELSE
	REPEAT
	update (eot);
	send_DT (next_send, eot);
	next_send:= next_send + 1; (2)
	UNTIL (eot OR (next_send = send_not_yet));
	rep_cnt:= 0;
	restart_tmo (SEND_TMO);
	END;
		SEND_CANC
		SEND

6.3.6.6.9 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 45.

Table 45 – 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
			DISC

		credit:= CC_credit;
		size:= decode (CC_pack_size);
		next_send:= 0;
		send_not_yet:= credit;
		send_data_or_cancel;
		END;
			SEND or
			SEND_CANC

	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 45 *(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;
			SEND or
			SEND_CANC

		ELSIF (expected = next_send) THEN close_send (AM_OK);	DISC

		ELSE
		REPEAT
		send_DT(expected, eot);
		expected := expected + 1; (2)
		UNTIL (expected = next_send);
		restart_tmo (SEND_TMO);
		rep_cnt := rep_cnt + 1;
		END;
			SEND

	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);	

	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
			SEND_CANC

Table 45 *(continued)*

Current state	Event	Action(s)	Next state (if<>current)

		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	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

6.3.6.6.10 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 46.

Table 46 – Consumer states and transitions

<table>
<tr><th>Current state</th><th>Event</th><th>Action(s)</th><th>Next state (if<>current)</th></tr>
<tr><td>(any)</td><td>tm_cancel_req</td><td>cancelled:= TRUE;</td><td></td></tr>
<tr><td>DISC</td><td>rcv_DR</td><td>send_DC (dc_reason);</td><td></td></tr>
<tr><td>DISC or
FROZEN</td><td>rcv_CR</td><td rowspan="2">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);
  IF (eot) THEN
    close_rcv (AM_OK);

  ELSE
    restart_tmo (ALIVE_TMO);
  END;

ELSE
  send_DR (err);
END;</td><td rowspan="2">LISTEN_
FROZEN

LISTEN

DISC</td></tr>
<tr><td>LISTEN_
FROZEN</td><td>rcv_CR AND
(CR_conn_ref <> conn_ref)</td></tr>
<tr><td>LISTEN_
FROZEN</td><td>rcv_CR AND
(CR_conn_ref = conn_ref)</td><td>send_CC
    (conn_ref, credit,
      min (CR_pack_size,
          my_pack_size);</td><td></td></tr>
<tr><td>FROZEN OR
LISTEN_
FROZEN</td><td>TMO</td><td></td><td>DISC</td></tr>
</table>

Table 46 *(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(dc_reason); 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 46 *(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

6.3.6.7 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. The following subclauses do not imply a particular implementation. Any interface which provides the same semantics is allowed.

6.3.6.7.1 Data exchange at the transport level

Figure 132 shows the interaction between transport layer and Session layer. The bullets indicate in which direction the parameters are passed.

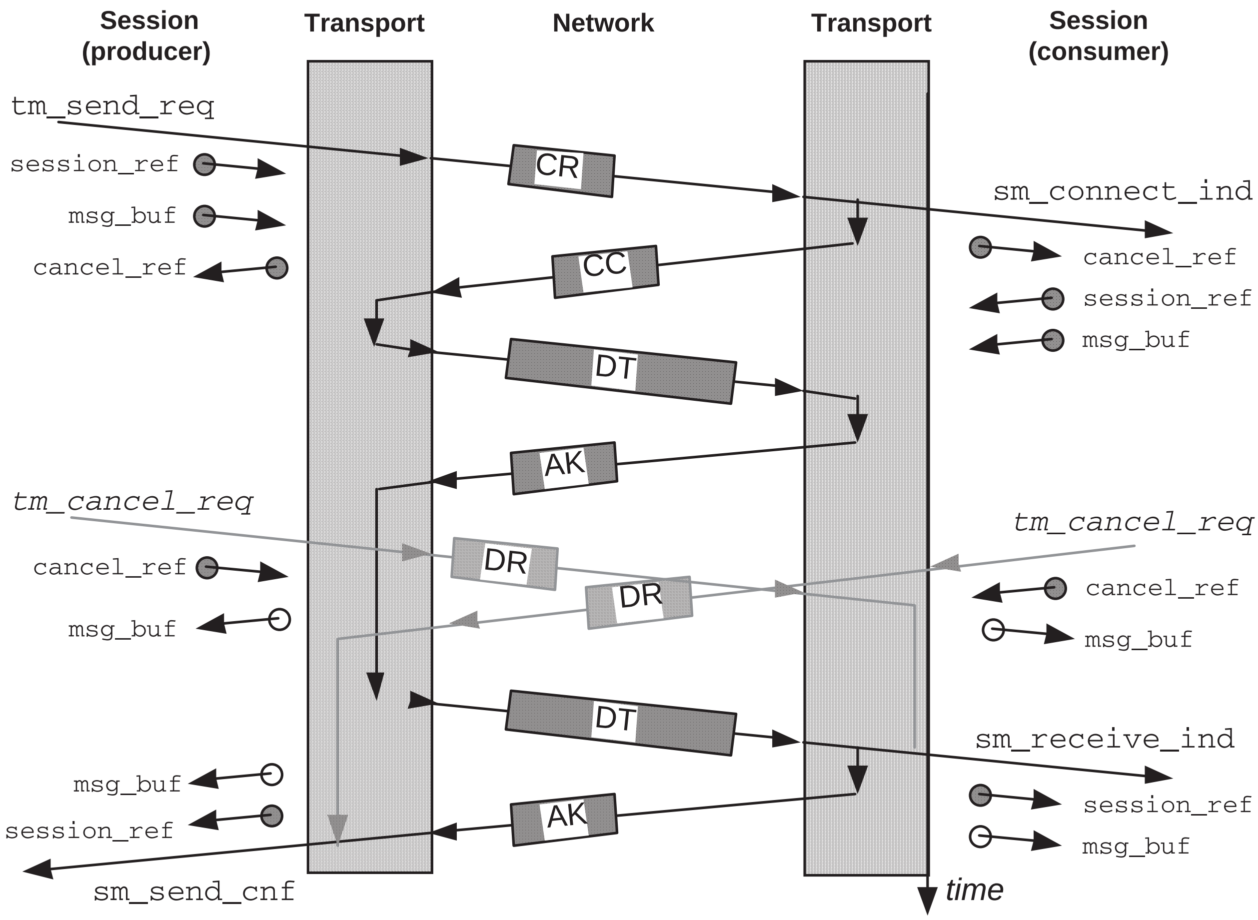


Figure 132 – Transport interface

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 call 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 47 and specified in the following subclauses.

Table 47 – 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

6.3.6.7.2 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.

6.3.6.7.3 Type 'TM_CONV_ID'

Definition: Type of a Conversation_Id

Syntax:

```
typedef struct     str_conv_id
                   {
UNSIGNED8          my_fct_or_station;
UNSIGNED8          node;
UNSIGNED8          function_or_station;
UNSIGNED8          next_station
                   } TM_CONV_ID;
```

6.3.6.7.4 Procedure 'tm_define_user'

Definition	Subscribes the indication procedures for a user identified as a Caller or as a Replier.	
Syntax	`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 );`	
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

6.3.6.7.5 Procedure ‘tm_send_req’

Definition	Requests transfer of a message	

Syntax	`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`
	`);`	

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.	

6.3.6.7.6 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	`typedef void ( * TM_SEND_CNF )`
	`(`
	`void * /* session_ref */`
	`UNSIGNED8 /* my_topo */`
	`AM_RESULT status`
	`);`	

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.	

6.3.6.7.7 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 /* in: conversation */
	TM_CONV_ID*, /* out: msg_addr */
	void * *, /* in: msg_len */
	UNSIGNED32, /* out hdr_addr*/
	void * *, /* out: hdr_len*/
	UNSIGNED *, /* in: cancel_ref */
	TM_MSG_DESCR *, /* out: session_ref */
	void * *, /* in: my_topo */
	UNSIGNED8, /* out: status */
	AM_RESULT);</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.	

6.3.6.7.8 Type ‘TM_RECEIVE_IND’

Definition	When it completely received an incoming message, the transport layer shall call the procedure sm_receive_ind of type TM_RECEIVE_IND.	

Syntax	`typedef void`
	`void *,`
	`AM_RESULT`
		`( * TM_RECEIVE_IND )`
		`(`
		`/* session_ref */`
		`/* status */`
		`);`

Input	session_ref	reference which couples sm_receive_ind with the previous sm_connect_ind.
	Status	AM_OK when the message is successfully received and placed into the buffer, otherwise another AM_RESULT error code.
Usage	1 – The Session layer is expected to supply a procedure ‘sm_receive_ind’ of type TM_RECEIVE_IND, in the ‘tm_define_user’ procedure. 2 – ‘sm_receive_ind’ is expected to return to the Session layer the buffers which the Session layer supplied in its ‘sm_connect_ind’.	

6.3.6.7.9 Procedure ‘tm_cancel_req’

Definition	Cancels a connection when called by either the Producer or the Consumer.	

Syntax	`AM_RESULT`
	`TM_MSG_DESCR *`
		`tm_cancel_req`
		`(`
		`cancel_ref`
		`);`

Input	cancel_ref	handle on the locked data structures of the transport layer.
Return		AM_OK if cancel applied to an ongoing message, AM_FAILURE otherwise.
Usage	1 – If the result is AM_OK, the transport layer will have sent a Disconnect_Request packet and will not have ‘sm_send_cnf’ or ‘sm_receive_ind’ will not have been called. 2 – If the message has been already completely sent and acknowledged, this procedure has no effect. 3 – ‘cancel_ref’ allows to release locked data structures, it is no more defined after a successful cancel operation.	

6.3.7 Multicast Transport Protocol (option)

6.3.7.1 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 133.

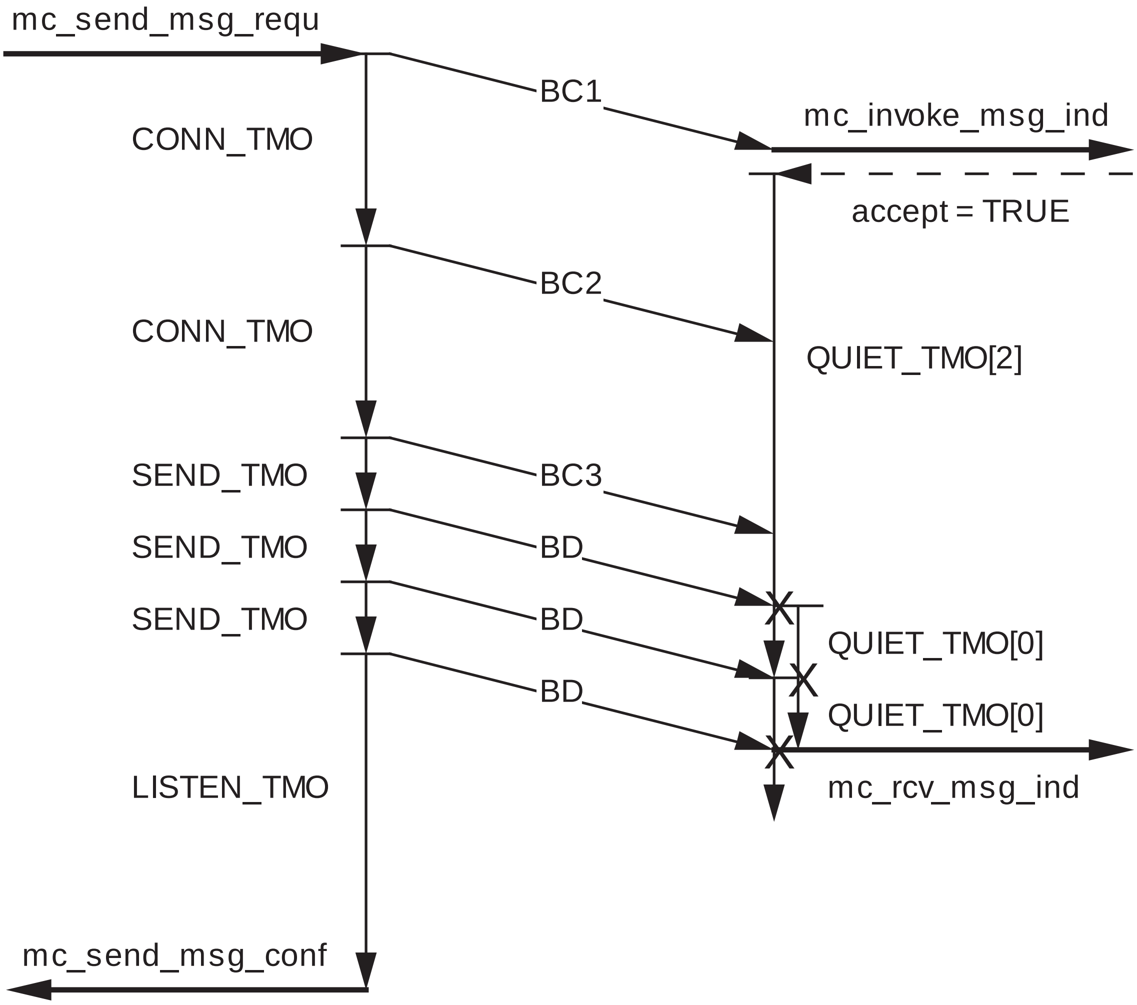


Figure 133 – Multicast message with no retransmission

6.3.7.1.1 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 134 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.

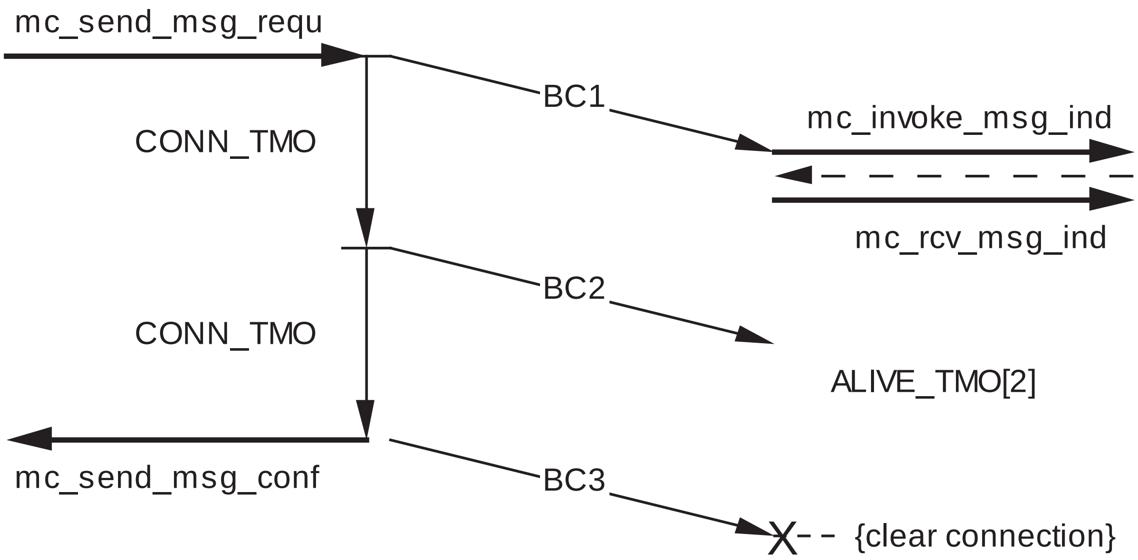


Figure 134 – Short multicast message with no BD packets and no loss

6.3.7.1.2 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 135.

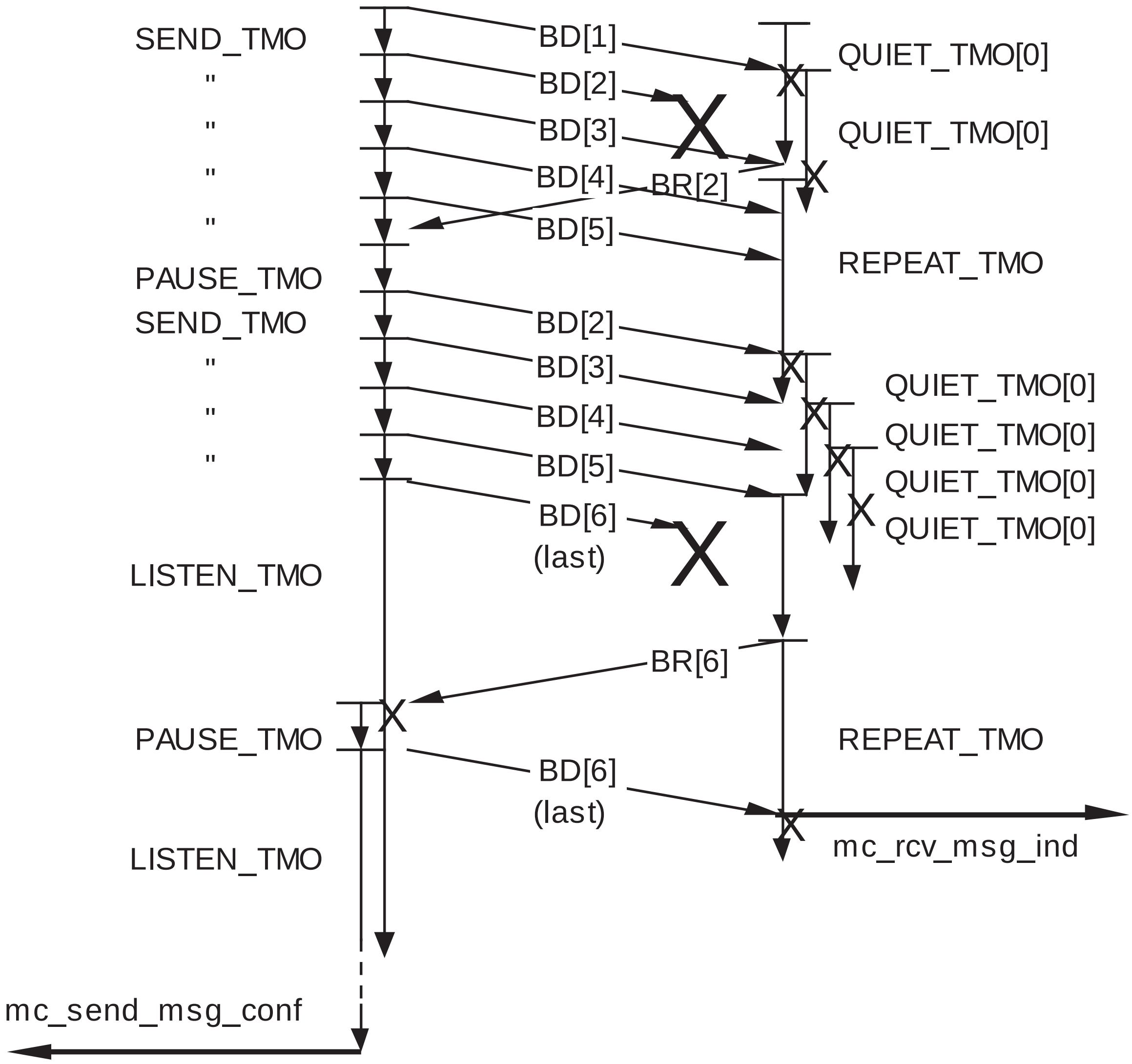


Figure 135 – Exchange with lost packets

6.3.7.1.3 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.

6.3.7.1.4 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.

6.3.7.1.5 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.When the table is 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. When 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).

6.3.7.1.6 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 ensures 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.

6.3.7.2 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.

6.3.7.3 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.

6.3.7.4 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 6.3.6.5.2, 'Message Transport Control Encoding'

6.3.7.5 Multicast Packets encoding

The Multicast Protocol distinguishes the following packets as shown in Figure 136.

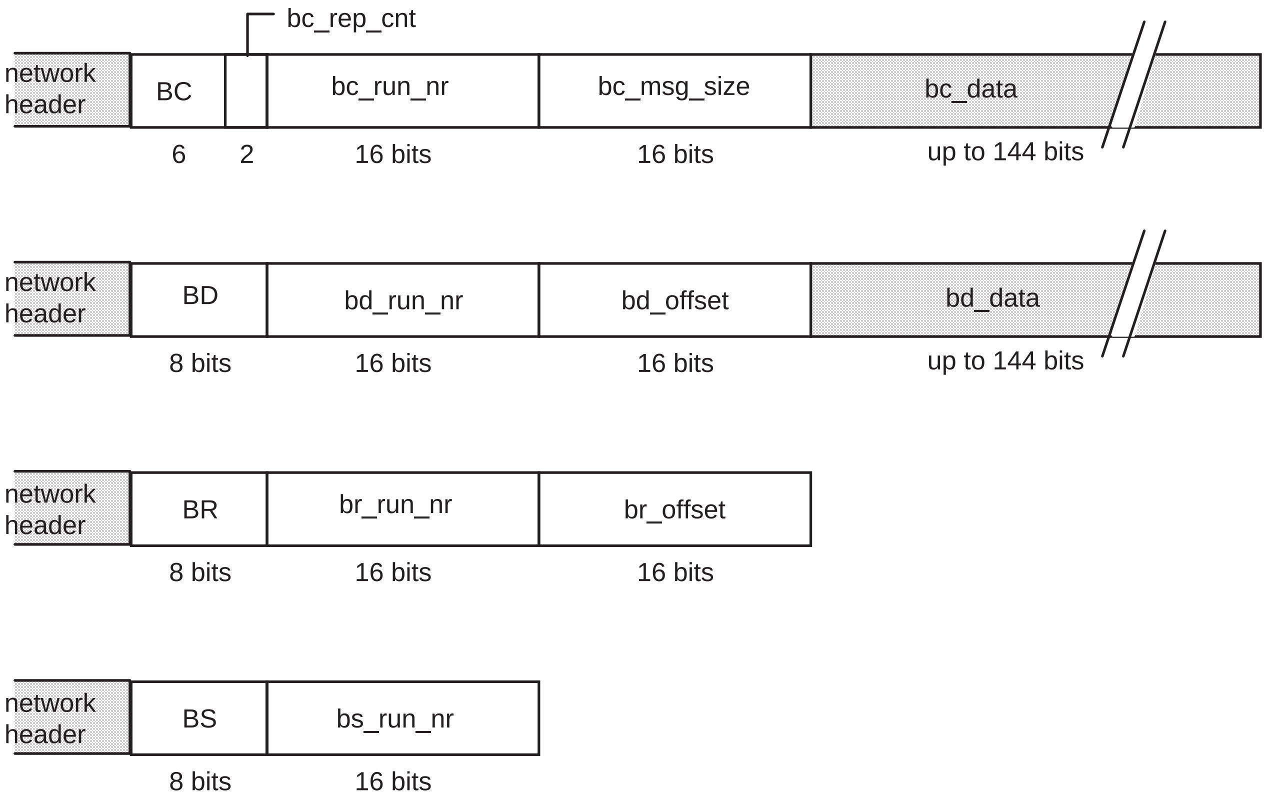


Figure 136 – Packet formats

Multicast Packets shall be encoded as specified in 6.3.6.5.4.

6.3.7.6 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.

6.3.7.6.1 States and state transition diagram

The multicast protocol machine, as illustrated in Figure 137, shall have the states listed in Table 48.

Table 48 – 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

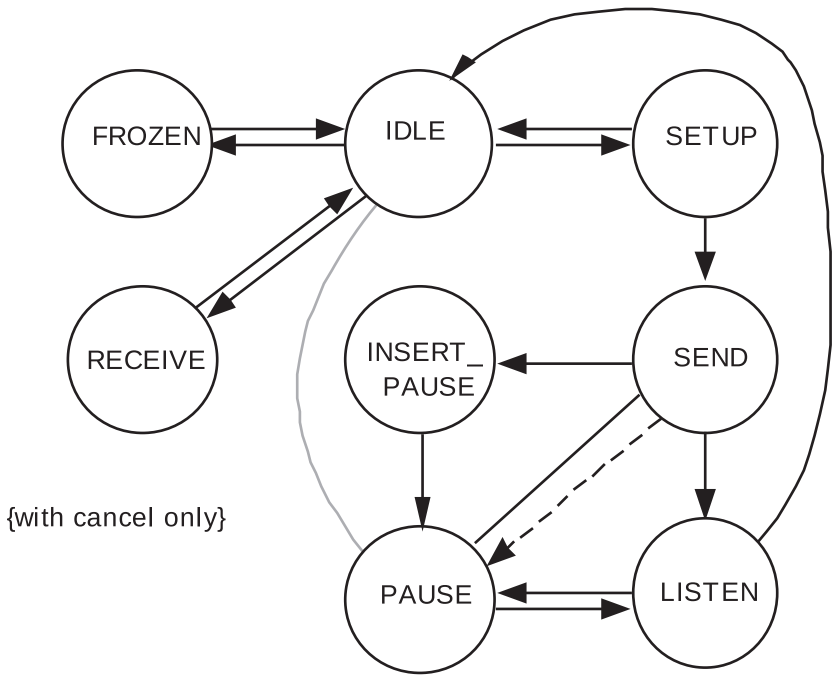


Figure 137 – Protocol machine states

The transitions between these states depend on incoming events defined in Table 49 and produce outgoing events defined in Table 50.

Table 49 – 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 50 – 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

6.3.7.6.2 Control fields in the data packet

The exchanged packets contain the control fields described in Table 51.

Table 51 – Control fields in packets

Event name	Field names	Short field description
xxx_BC (	BC_run_nr,	Connection_Reference
	BC_rep_cnt,	count of pending BC retransmissions
	BC_msg_size)	total message size
xxx_BD (	BD_run_nr,	Connection_Reference
	BD_offset)	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.		

6.3.7.6.3 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 52 shall be implemented.

Table 52 – 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.		

6.3.7.6.4 Constants

Table 53 lists the constants used in the Multicast protocol machine.

Table 53 – 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 × 64 [ms]	processing time for events (polling period)
NET_DELAY	8 × 64 [ms]	expected transmission delay (end-to-end)
REP_RANGE	LISTEN_TMO / SEND_TMO × BD_DATA_SIZE	
NOTE Offset range within which retransmission is possible defines the window size REP_RANGE.		

The time-outs shall be as specified in Table 54.

Table 54 – MCP time-outs

Time-out name	Time-out value
	Producer site
CONN_TMO	4 × 64 [ms]
SEND_TMO	1 × 64 [ms]
PAUSE_TMO	1 × 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) × (REPEAT_TMO + PROC_DELAY) + 2 × NET_DELAY + PROC_DELAY – SEND_TMO
	Consumer site
QUIET_TMO [MAX_TRIALS]	SEND_MAX_TMO + PROC_DELAY + NET_DELAY + ix × (CONN_TMO + PROC_DELAY)
ALIVE_TMO [MAX_TRIALS]	NET_DELAY + PACK_LIFE_TIME + ix × (CONN_TMO + PROC_DELAY)
REPEAT_TMO	SEND_MAX_TMO + 2 × 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)’.	

6.3.7.6.5 Compound actions

The compound actions listed in Table 55 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 55 – 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;
		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 56.

Table 56 – Filtering of BR packets

Action name	Action specification

init_BR_filter	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;

check_BR (VAR accept)	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;

6.3.7.6.6 Producer state event table

Table 57 specifies the behaviour of the Producer.

Table 57 – 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
			
			IDLE
			SEND

SEND	rcv_BR	check_BR (VAR accept);
		IF (accept) THEN
		
		END;
			INSERT_
			PAUSE

	TMO AND cancelled	restart_tmo (PAUSE_TMO);	PAUSE

Table 57 *(continued)*

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

6.3.7.6.7 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 58.

Table 58 – MCP Consumer state event table

Current state	Event	Action(s)	Next state
(any)	mc_cancel_msg	cancelled:= TRUE;	
IDLE	rcv_BC	accept_BC;	IDLE OR FROZEN OR RECEIVE

RECEIVE	rcv_BD AND
	(BD_run_nr = conn_run_nr) AND
	(my_node = final_node)
		IF (cancelled) THEN
		reset_tmo;
		ELSIF (BD_offset = offset) THEN
		update (offset);
		IF (offset = msg_size) THEN
		close_rcv (AM_OK);
		ELSE
		rep_cnt:= 0;
		restart_tmo (QUIET_TMO[0]);
		END;
		ELSIF (BD_offset > offset) THEN
		IF (rep_cnt = 0) THEN
		send_BR (conn_run_nr, offset);
		INC (rep_cnt);
		restart_tmo (REPEAT_TMO);
		END;
		ELSE
		rep_cnt:= 0;
		restart_tmo (QUIET_TMO[0]);
		END;
			
			IDLE
			
			IDLE

	TMO AND
	(rep_cnt < (MAX_TRIALS – 1))
		IF (cancelled) THEN
		reset_tmo;
		ELSE
		send_BR (conn_run_nr, offset);
		INC (rep_cnt);
		restart_tmo (REPEAT_TMO);
		END;
			
			IDLE

	TMO AND (rep_cnt = (MAX_TRIALS – 1))	close_rcv (AM_REPEAT_TMO_ERR);	IDLE
	rcv_BC AND (BC_run_nr = conn_run_nr) AND (my_node = final_node)	(* no action *)	

Table 58 (*continued*)

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

6.3.8 Message session layer

6.3.8.1 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 138.

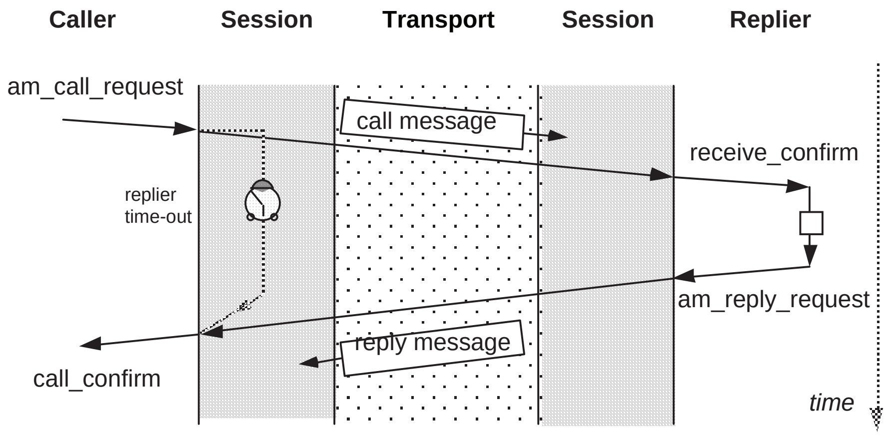


Figure 138 – Session layer transfer

The Session layer uses the services of the transport layer for each message transfer.

6.3.8.2 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.

6.3.8.3 Shortcut

The Session layer shall shortcut the network when the partner resides on the same station, i.e. forward the message directly without involving the transport layer.

6.3.8.4 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.

6.3.8.5 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 139.

In the Connect_Request of a Reply_Message, the Session_Header consists of one octet containing the reply status supplied by the Replier application.

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

Figure 139 – Session_Header in Call_Message (of type Am_Result)

6.3.8.6 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.

6.3.8.7 Session layer interface

The interface of the Session layer is identical to that of the Application layer since the presentation layer has no protocols.

6.3.9 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 6.4.

6.3.10 Message Application layer

6.3.10.1 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).

6.3.10.2 Application_Messages_Interface

6.3.10.2.1 AMI primitives

The Application Layer Interface shall implement the primitives illustrated in Figure 140, listed in Table 59 and specified in the following subclauses.

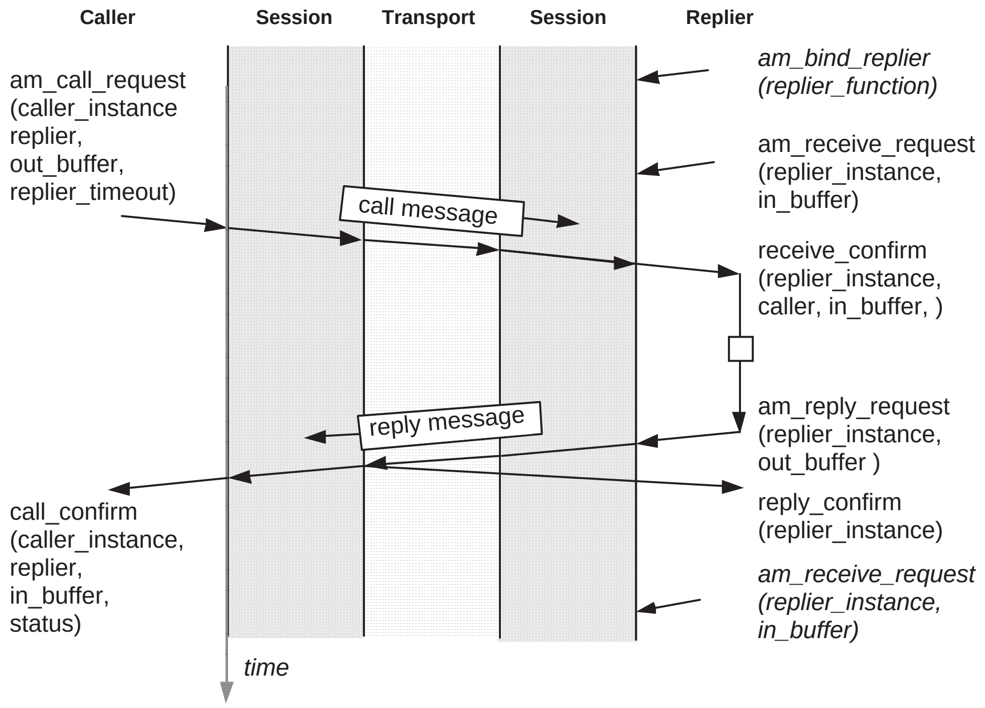


Figure 140 – Application_Messages_Interface

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;

OVF - overflow;

TMO - time-out.

Table 59 – 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.

6.3.10.2.2 Definition of AM_RESULT

<table>
<tr><td>Definition</td><td>A procedure which returns a value of type AP_RESULT shall encode it as follows:</td></tr>
<tr><td>Syntax</td><td><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></td></tr>
</table>

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.

6.3.10.2.3 Address constants

The constants listed in Table 60 are reserved identifiers.

Table 60 – 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.

6.3.10.2.4 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	`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;`	

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 below.

A convenient encoding of an application address is shown in Figure 141:

7	6	5	4	3	2	1	0
snu	gni	node_or_group					
func_or_stat							
next_station							
trv	tpv	topo_counter					

Figure 141 – Encoding of AM_ADDRESS

NOTE AM_ADDRESS is an interface format, Am_Address is transmission format.

6.3.10.3 Caller side

6.3.10.3.1 Own identification

A Caller shall identify itself by its Function_Id.

NOTE The Caller identifies itself in 'am_call_request'.

6.3.10.3.2 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'.

6.3.10.3.3 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).

6.3.10.3.4 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.

6.3.10.3.5 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 reflects the message back over the consist network.

6.3.10.3.6 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.

6.3.10.3.7 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. If shall be computed as follows:

a) if Next_Station is specified (Next_Station_Id <> AM_UNKNOWN), the Link_Address shall be taken from the station directory, using Next_Station_Id as entry;

b) if 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 <> 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.

6.3.10.3.8 Topo_Counter

Bit 7 (most significant) of this octet shall be 0.

Bit 6 of this octet shall be 0.

Bits 0 to 5 shall contain a valid Topo_Counter in the range from 1 to 63.

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 <> 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.

6.3.10.3.9 Use of Network_Address at the Caller

The System_Address and User_Address modes are summarised in Table 61.

Table 61 – 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 (fundi(AM_ROUTER_FCT)) (Agent on remote Node)
	<> AM_SAME_STATION and <> AM_UNKNOWN	stadi (Station_Id)	stadi(fundi(AM_ROUTER_FCT)) (Agent on remote station)
	AM_UNKNOWN	error	stadi(fundi(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(fundi(AM_ROUTER_FCT))
AM_UNKNOWN	any	stadi (fundi(Function_Id)) (error if function not registered)	stadi(fundi(AM_ROUTER_FCT)) or (train bus, Node_Address)

6.3.10.4 Replier site

6.3.10.4.1 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.

6.3.10.4.2 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.

6.3.10.4.3 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.

6.3.10.4.4 Group or Individual (gni)

The 'gni' bit shall be set to 1 to indicate 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.

6.3.10.4.5 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.

6.3.10.4.6 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.

6.3.10.4.7 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.

6.3.10.5 Initialisation

The Message Service is initialised at different levels by the following procedures.

The following subclauses do not imply a particular implementation. Any interface which provides the same semantics is allowed.

6.3.10.5.1 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.	

6.3.10.5.2 Procedure 'am_announce_device'

Definition	Announces the device's configuration.	

Syntax	`AM_RESULT`
	
	`UNSIGNED16`
	`UNSIGNED16`
	`UNSIGNED16`
	`UNSIGNED8`
		`am_announce_device`
		`(`
		`max_call_number,`
		`max_inst_number,`
		`default_reply_timeout,`
		`my_credit`
		`);`

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.	

6.3.10.5.3 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.

6.3.10.5.4 Procedure 'am_set_current_tc'

Definition	Sets the current value of the Topo_Counter 'this_topo' for the network layer.	
Syntax	`AM_RESULT`	`am_set_current_tc`
		`(`
	`UNSIGNED8`	`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.	

6.3.10.6 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.

The following subclauses do not imply a particular implementation. Any interface which provides the same semantics is allowed.

6.3.10.6.1 Type ‘AM_STADI_ENTRY’

Definition	Type of a station directory entry	

Syntax	`typedef struct`
	`{`
	`UNSIGNED8 station;`
	`UNSIGNED8 next_station;`
	`ENUM8 bus_id;`
	`UNSIGNED64 device_adr;`
	`} AM_STADI_ENTRY;`	

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)

6.3.10.6.2 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	`AM_RESULT am_stadi_write`
	`(`
	`const AM_STADI_ENTRY entries[],`
	`UNSIGNED8 nr_of_entries`
	`);`	

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.

6.3.10.6.3 Procedure ‘am_stadi_read’

Definition	Reads a number of entries from the station directory.	

Syntax	`AM_RESULT am_stadi_read`
	`(`
	`AM_STADI_ENTRY entries[],`
	`UNSIGNED8 nr_of_entries`
	`);`	

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.

6.3.10.7 Function directory interface

The following subclauses do not imply a particular implementation. Any interface which provides the same semantics is allowed.

6.3.10.7.1 Type 'AM_DIR_ENTRY'

Definition	Type of a function directory entry.	
Syntax	`typedef struct AM_DIR_ENTRY { UNSIGNED8 function; UNSIGNED8 station; } AM_DIR_ENTRY;`	

```
typedef struct     AM_DIR_ENTRY
                   {
    UNSIGNED8      function;
    UNSIGNED8      station;
                   } AM_DIR_ENTRY;
```

6.3.10.7.2 Procedure 'am_clear_dir'

Definition	Sets the Station_Id of all functions in the directory to AM_UNKNOWN.	
Syntax	`AM_RESULT am_clear_dir (void);`	
Output		AM_OK, AM_FAILURE

6.3.10.7.3 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	`AM_RESULT am_insert_dir_entries ( AM_DIR_ENTRY * function_list, unsigned number_of_entries );`	
Input	function_list	function directory list
	number_of_entries	number of elements in this list
Return		AM_OK, AM_FAILURE

```
AM_RESULT          am_insert_dir_entries
                   (
    AM_DIR_ENTRY * function_list,
    unsigned       number_of_entries
                   );
```

6.3.10.7.4 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	`AM_RESULT am_remove_dir_entries ( AM_DIR_ENTRY * function_list, unsigned number_of_entries );`	
Input	function_list	function directory list
	number_of_entries	number of elements in this list
Return		AM_OK, AM_FAILURE

```
AM_RESULT          am_remove_dir_entries
                   (
    AM_DIR_ENTRY * function_list,
    unsigned       number_of_entries
                   );
```

6.3.10.7.5 Procedure 'am_get_dir_entry'

Definition	Reads the Station_Id for a given Function_Id from the function directory.	
Syntax	`AM_RESULT am_get_dir_entry ( UNSIGNED8 function, UNSIGNED8 * station );`	
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

6.3.10.8 Group directory interface

The following subclauses do not imply a particular implementation. Any interface which provides the same semantics is allowed.

6.3.10.8.1 Type AM_GROUP

Definition	Type of an entry in the group directory
Syntax	`typedef UNSIGNED8 AM_GROUP;`
Usage	Groups are identified by a 6-bit address, represented by one octet, the two most significant bits being ignored.

6.3.10.8.2 Procedure 'am_clear_groups'

Definition	Clears all entries in the group directory.	
Syntax	`AM_RESULT am_clear_groups (void);`	
Return		AM_OK, AM_FAILURE

6.3.10.8.3 Procedure 'am_insert_member'

Definition	Registers this Station_Id as a member into the group directory.	
Syntax	`AM_RESULT am_insert_member ( AM_GROUP group );`	
Input	group	Group to which this station is to pertain
Return		AM_OK, AM_FAILURE

6.3.10.8.4 Procedure 'am_remove_member'

Definition	Removes this Station_Id as a Group member.	

Syntax	`AM_RESULT am_remove_member (`
	`AM_GROUP group`
	`);`	

Input	group	Group from which this station is to be removed.
Return		AM_OK, AM_FAILURE

6.3.10.8.5 Procedure 'am_member'

Definition	Checks if the specified Station_Id is a member of a Group.	

Syntax	`AM_RESULT am_member (`
	`AM_GROUP group`
	`);`	

Input	group	Group to which this Station_Id pertains.
Return		AM_OK if the Station_Id is a member of the Group, AM_FAILURE otherwise.

6.3.10.9 Caller application interface

The following subclauses do not imply a particular implementation. Any interface which provides the same semantics is allowed.

6.3.10.9.1 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	`void` `UNSIGNED8` `const AM_ADDRESS *` `void *` `UNSIGNED32` `void *` `UNSIGNED32` `UNSIGNED16` `AM_CALL_CONFIRM` `void *`	`am_call_request (` `caller_function,` `replier,` `out_msg_adr,` `out_msg_size,` `in_msg_adr,` `in_msg_size,` `reply_timeout,` `call_confirm,` `caller_ref` `);`
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.	

6.3.10.9.2 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	`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`
	`);`	

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.	

6.3.10.9.3 Procedure ‘am_call_cancel’

Definition	Cancels a call request which is not yet confirmed.	

Syntax	`AM_RESULT` `am_call_cancel`
	`(`
	`UNSIGNED8` `caller_function,`
	`const AM_ADDRESS *` `replier`
	`);`	

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.	

6.3.10.10 Replier application interface

The following subclauses do not imply a particular implementation. Any interface which provides the same semantics is allowed.

6.3.10.10.1 Procedure ‘am_bind_replier’

Definition	Makes a Replier known to the Messenger and connects its procedures.	

Syntax	`AM_RESULT`
	`UNSIGNED8`
	`AM_RECEIVE_CONFIRM`
	`AM_REPLY_CONFIRM`
		`am_bind_replier`
		`(`
		`replier_function,`
		`receive_confirm,`
		`reply_confirm`
		`);`

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’.	

6.3.10.10.2 Procedure ‘am_unbind_replier’

Definition	Cancels all instances of the specified Replier and removes the binding.	

Syntax	`AM_RESULT`
	`UNSIGNED8`
		`am_unbind_replier`
		`(`
		`replier_function`
		`);`

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.	

6.3.10.10.3 Procedure 'am_receive_request'

Definition	Informs that a Replier is ready to receive an incoming call.	
Syntax	`AM_RESULT am_receive_request ( UNSIGNED8 replier_function, void * in_msg_adr, UNSIGNED32 in_msg_size, void * replier_ref );`	
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 were 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.	

6.3.10.10.4 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	`typedef void ( * AM_RECEIVE_CONFIRM ) (`
	`UNSIGNED8 replier_function,`
	`const AM_ADDRESS caller,`
	`* in_msg_adr ,`
	`void * in_msg_size,`
	`UNSIGNED32 replier_ref`
	`void * );`	

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'.	

6.3.10.10.5 Procedure ‘am_reply_request’

Definition	Requests to send a Reply_Message in response to a previously received Call_Message	

Syntax	`AM_RESULT am_reply_request`
	`(`
	`UNSIGNED8 replier_function,`
	`void * out_msg_adr,`
	`UNSIGNED32 out_msg_size,`
	`void * replier_ref`
	`AM_RESULT status);`	

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.	

6.3.10.10.6 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	`typedef void ( * AM_REPLY_CONFIRM )`
	`(`
	`UNSIGNED8 replier_function,`
	`void * replier_ref`
	`);`	

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.	

6.3.10.10.7 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	`AM_RESULT am_receive_cancel ( UNSIGNED8 replier_function, void * replier_ref );`	
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 deallocate dynamic buffers for the Call_Messages.	

6.3.10.11 Procedure ‘am_buffer_free’

Definition	Deallocates a message buffer previously allocated by the Session layer after use.	
Syntax	`AM_RESULT am_buffer_free ( void * in_msg_adr, UNSIGNED32 size );`	
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.	

6.3.10.12 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 deallocate a dynamic buffer if such is used. No Reply_Message, however, shall be generated in this case.

6.4 Presentation and encoding of transmitted and stored data

6.4.1 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.

6.4.2 Data ordering

6.4.2.1 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.

6.4.2.2 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.

6.4.2.3 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 these used for storage or transmission and vice-versa.

6.4.2.4 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 least significant bit of that integer would have offset zero. This bit is considered to be the 'rightmost' one when reading the data structure.

d) To improve comprehensibility, the bit numbering in the following clauses 6.4.3.x is aligned with a programmers view to the traffic store data structure. So a bit offset x can be interpreted as the x^{th} power of two 2^x. The x^{th} bit in a bitset can be computed as an octet offset and a bit offset as follows: octet offset = x divided by 8 rounded down, bit offset (inside the octet) = x modulo 8.

e) All data shall be transmitted most significant octet first (Big-Endian)

f) 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.

g) 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.

h) The elements of a structured type (Record, Sequence) shall be transmitted in the order they are declared.

i) 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).

j) To ease implementation, a variable shall be stored at an offset address which is a multiple of its size (alignment).

k) Variable length data (open arrays, records, sets, etc.) shall not be used as Process_Variables, but may be transmitted in messages.

6.4.2.5 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.

6.4.3 Notation for the primitive types

6.4.3.1 Notation for the boolean type

6.4.3.1.1 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.).

6.4.3.1.2 Syntax

```
BooleanType::= BOOLEAN1
```

6.4.3.1.3 Encoding

A variable of boolean type shall be encoded as one bit:

1st	interpretation
0	FALSE
1	TRUE

6.4.3.2 Notation for the antivalent type

6.4.3.2.1 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 variables for other variables or for critical Booleans.

6.4.3.2.2 Syntax

```
AntivalentType::= ANTIVALENT2
```

6.4.3.2.3 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:

1	0	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.

6.4.3.3 Notation for the unsigned integer types

6.4.3.3.1 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.

6.4.3.3.2 Syntax

```
UnsignedType::= UNSIGNED#, (# = any unsigned integer).
```

6.4.3.3.3 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.

6.4.3.3.3.1 UNSIGNED8 encoding

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Range: 0..255

6.4.3.3.3.2 UNSIGNED16 encoding

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

6.4.3.3.3.3 UNSIGNED32 encoding

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

Range: $0..+2^{32}–1$

6.4.3.4 Notation for the integer type

6.4.3.4.1 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 an ASN.1 ‘integer type’, restricted to a fixed size of #.

6.4.3.4.2 Syntax

```
IntegerType::= INTEGER#, (# = any unsigned integer).
```

6.4.3.4.3 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’).

6.4.3.4.3.1 INTEGER8 encoding

7	6	5	4	3	2	1	0
sign	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Range: –128 .. +127

EXAMPLE ‘1111 1110’B = – 2

6.4.3.4.3.2 INTEGER16 encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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: $-2^{15} .. 2^{15} - 1$

6.4.3.4.3.3 INTEGER32 encoding

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

Range: $-2^{31} .. +2^{31} - 1$

6.4.3.5 Notation for the enumerated type

6.4.3.5.1 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 #.

6.4.3.5.2 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’.

6.4.3.5.3 Encoding

Values of ENUM# shall be represented by an unsigned integer occupying the same place.

6.4.3.5.3.1 ENUM4 encoding

3	2	1	0
2^3	2^2	2^1	2^0

Range: 0..15

EXAMPLE ‘0001’B means ‘Monday’ in the above example.

6.4.3.5.3.2 ENUM8 encoding

7	6	5	4	3	2	1	0
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).

6.4.3.6 Notation for the binary coded decimal type

6.4.3.6.1 Definition

A 4-bit unsigned integer expressing a decimal digit between 0 and 9.

NOTE This type does not exist in ASN.1.

6.4.3.6.2 Syntax

```
BinaryCodedDecimalType::= BCD4
```

6.4.3.6.3 Encoding

A BCD4 shall be encoded as an unsigned integer occupying the same place.

3	2	1	0
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.

6.4.3.7 Notation for the unipolar types

6.4.3.7.1 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’.

6.4.3.7.2 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 (double byte).

6.4.3.7.3 Encoding

A variable of unipolar type shall be transmitted as an unsigned integer.

6.4.3.7.3.1 UNIPOLAR2.16 encoding

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

6.4.3.8 Notation for the bipolar types

6.4.3.8.1 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’.

6.4.3.8.2 Syntax

```
BipolarType::= BIPOLAR2.16 | BIPOLAR4.16
```

NOTE 1 The number before the dot 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 (double byte).

6.4.3.8.3 Encoding

6.4.3.8.3.1 BIPOLAR2.16 encoding

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

6.4.3.8.3.2 BIPOLAR4.16 encoding

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

6.4.3.9 Notation for the real type

6.4.3.9.1 Definition

A primitive type whose distinguished values are members of the set of real numbers.

6.4.3.9.2 Syntax

```
RealType::= REAL32
```

6.4.3.9.3 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.

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

Range: $\pm$ 3,37 $\times$ 10^{+38}

6.4.3.10 Notation for the character type

6.4.3.10.1 Definition

A primitive type whose distinguished values are members of the set of characters defined in ISO/IEC 8859-1.

6.4.3.10.2 Syntax

```
CharacterType::= CHARACTER8
```

6.4.3.10.3 Encoding

Characters shall be transmitted in one octet, without parity bit.

7	6	5	4	3	2	1	0
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.

6.4.3.11 Notation for the Unicode character type

6.4.3.11.1 Definition

A primitive type whose distinguished values are members of the set of characters defined in ISO/IEC 10646.

6.4.3.11.2 Syntax

```
UnicodeType::= UNICODE16
```

6.4.3.11.3 Encoding

Unicode characters shall be transmitted in two octets.

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

6.4.3.12 Notation for the uncommitted types

6.4.3.12.1 Definition

An uncommitted type of undefined contents, but of fixed size.

6.4.3.12.2 Syntax

```
AnyType::= WORD#, (# = any unsigned integer)
```

6.4.3.12.3 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.

6.4.3.12.3.1 WORD8 encoding

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

6.4.3.12.3.2 WORD16 encoding

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

6.4.4 Structured types

6.4.4.1 General

Five different structured types are defined:

a) RECORD (variable length),

b) ARRAY (fixed length or variable length),

c) BITSET# (fixed length),

d) ONE_OF (variable length)

e) SOME_OF (variable length).

6.4.4.2 Notation for the record types

6.4.4.2.1 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.

6.4.4.2.2 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

6.4.4.2.3 Encoding

Elements of a RECORD shall be transmitted in the order of their declaration.

EXAMPLE A value of type Date32 is represented as follows:

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

6.4.4.3 Notation for the bitset types

6.4.4.3.1 Definition

An ARRAY [#] of BOOLEAN1, having a fixed size in bits defined by the postfix #.

NOTE This type corresponds to BITSTRING in ASN.1.

6.4.4.3.2 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.

6.4.4.3.3 Encoding

Elements of a bitset shall be transmitted in order of declaration.

6.4.4.3.3.1 BITSET8 encoding

7	6	5	4	3	2	1	0
8th							1st

Range: 8-bit Set of Boolean

EXAMPLE
```
AccessType8::= BITSET8
  {
  system     (0),     -- first bit of the bitset (LSB)
  owner      (1),
  group      (2),
  world      (3),
  }
```

is equivalent to

```
AccessType8::= BITSET8
  {
  system,             -- first bit of the bitset (LSB)
  owner,
  group,
  world,
  reserved4
  reserved5
  reserved6
  reserved7           -- 8th or last bit of the bitset (MSB)
  }
```

An UNSIGNED8 occupying that space with a value of ‘01’H means that ‘system’ is the only member of the set.

6.4.4.3.3.2 BITSET16 encoding

8th							1st	16th							9th

EXAMPLE
```
AccessType::= BITSET16 { system (0), owner (1), group (2), world (3)}
```

Value ‘0000 0000 0000 0110’B means that ‘owner’ and ‘group’ are members of the set.

6.4.4.3.3.3 BITSET32 encoding

8th							1st	16th							9th
24th							17th	32nd							25th

6.4.4.3.3.4 BITSET64 encoding

8th							1st	16th							9th
24th 40th							17th 33th	32nd 48th							25th 41st
56th							49th	64th							57th

6.4.4.4 Notation for the array type

6.4.4.4.1 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).

6.4.4.4.2 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 an unsigned type with a defined identifier is indicated, this declares the corresponding field.

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.

If a stop value is defined to close an open array, the value shall be of the same type as the array element.

The size may be given by an arithmetic expression.

6.4.4.4.3 Encoding

Arrays shall be transmitted in order of increasing index.

Multi-dimensional arrays shall be transmitted in the order their indices are listed.

NOTE ARRAY OF [row, column] is transmitted row by row.

Arrays of octets (uncommitted contents, e.g. memory dump) shall be transmitted in increasing memory address (or index) of the Application Memory.

All elements of the array shall be transmitted, even those which are not significant.

EXAMPLE 1 Transmission of an octet memory dump:

```
DumpOctetType::= ARRAY [octet_count UNSIGNED16] OF WORD8.
```

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="16">octet_count</td></tr>
<tr><td colspan="8">octet at address M</td><td colspan="8">octet at address M+1</td></tr>
<tr><td colspan="16">octets</td></tr>
<tr><td colspan="8">octet at address (M + octet_count-2)</td><td colspan="8">octet at address (M + octet_count-1)</td></tr>
</table>

EXAMPLE 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.
```

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="16">word_count</td></tr>
<tr><td colspan="16">word at address M</td></tr>
<tr><td colspan="16"></td></tr>
<tr><td colspan="16">word at address (M + word_count × 2 –2)</td></tr>
</table>

EXAMPLE 3 Number of elements given by a field in the nesting data structure (at an unspecified offset):

```
DumpOctetType::= ARRAY [array_count] OF WORD8.
```

EXAMPLE 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
  }
```

EXAMPLE 5 Character strings, in which the stop character is a ‘space’:

```
ProfibusString::= ARRAY [STOP = ‘20’H] OF CHARACTER8.
```

6.4.4.5 Notation for the choice types

6.4.4.5.1 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.

6.4.4.5.2 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.

EXAMPLE 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
  }
```

EXAMPLE 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
  }
```

EXAMPLE 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;
  ...
  }
```

EXAMPLE 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.

6.4.4.5.3 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:

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="16">choice_var (= 5)</td></tr>
<tr><td colspan="8">first octet of (StandbySequence)</td><td colspan="8"></td></tr>
<tr><td colspan="8"></td><td colspan="8">last octet of (StandbySequence) or dummy</td></tr>
</table>

6.4.4.6 Notation for the set types

6.4.4.6.1 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.

6.4.4.6.2 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 is an enumerated type, the enumeration constants used to select the set elements shall be put in brackets.

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.

EXAMPLE 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
  }
```

EXAMPLE 2 Omission of the reference name

```
MemberType::= SOME_OF [UNSIGNED8]
  {
  [3]                  Type_OpenSequence,
  [2]                  Type_CloseSequence,
  [5]                  Type_StandbySequence
  }
```

EXAMPLE 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
 }
```

EXAMPLE 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
  ...
  }
```

6.4.4.6.3 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.

EXAMPLE 1 A particular value of the above MemberType set will be transmitted as:

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">member = ‘02’H</td><td colspan="8">closesequence</td></tr>
<tr><td colspan="16">..
closesequence
..</td></tr>
<tr><td colspan="8">closesequence (end)</td><td colspan="8">member =‘05’H</td></tr>
<tr><td colspan="16">..
standbysequence
..</td></tr>
<tr><td colspan="8">Standbysequence</td><td colspan="8">member = ‘FF’H</td></tr>
</table>

EXAMPLE 2 If the tag is replaced by a bitset, the coding will be:

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

6.4.5 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.
```

6.4.6 Notation for special types

Some structured types have a special type designator.

6.4.6.1 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 STRING32is represented by an ARRAY [32 STOP=‘00’H] OF CHARACTER8.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1st character or ‘00’H								2nd character or ‘00’H							
characters ...															
last character or ‘00’H								32nd character or ‘00’H							

6.4.6.2 Notation for the TIMEDATE48 type

6.4.6.2.1 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.

6.4.6.2.2 Syntax

```
TimeDate48::= RECORD
  {
  seconds            SIGNED32,         -- elapsed since 1970, January
                                          1st, 00:00
  ticks              UNSIGNED16        -- fraction of seconds (1 tick
                                          = 1/65536s)
  }
```

6.4.6.2.3 Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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 68 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.

6.4.6.3 Notation for the TIME64 type

6.4.6.3.1 Definition

A structured type expressing the absolute time (UTC) in seconds since 1900, January 1st, 00:00. This time is not compensated by leap seconds.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
seconds (most significant)															
seconds (least significant)															
ticks = 1/65536 s															
chirp = 0,232 μs															

NOTES 1 This time definition is taken from Internet RFC1305, which defines the synchronisation protocol for a distributed clock systems. It is different from UNIX time, which is based on the year 1970.

NOTE 2 A Time64 variable will wrap around in the year 2036. This wrap-around should be considered in the test of the software. This time definition can therefore also be considered as defining the time remaining until January 2036.

6.4.6.4 Notation for the ASN.1 boolean8 type

6.4.6.4.1 Definition

A primitive type with two distinguished values, TRUE and FALSE.

NOTE This is the ASN.1 ‘BooleanType’.

6.4.6.4.2 Syntax

```
Boolean8Type::= BOOLEAN8
```

6.4.6.4.3 Encoding

A variable of boolean 8 type shall be encoded on 8 bits, ‘00000000’B being interpreted as FALSE and any other value as TRUE.

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	FALSE
0	0	0	0	0	0	0	1	TRUE

7 Application Layer

7.1 Process Data Marshalling

7.1.1 Marshalling Types

PDM copies process variables from one Traffic Store to another Traffic Store:

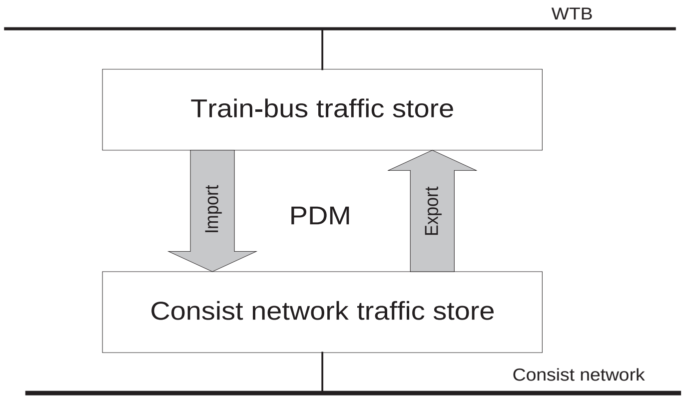


Figure 142 – Process Data Marshalling

Two types of marshalling are defined:

- Export Marshalling
- Import Marshalling

7.1.1.1 Export Marshalling

Export Marshalling means, to copy variables from one or more Consist network Traffic Stores to the WTB Traffic Store source port. The entire WTB port is written, so unused space in the port has to be filled with default values. The Export Marshalling can do some processing on the process variables.

The Export Marshalling determines the length of the exported frame depending on the frame type.

7.1.1.2 Import Marshalling

Import Marshalling means, to copy variables from the WTB Traffic Store to the statically configured Consist network Traffic Store(s). The Import Marshalling can do some processing on the process variables.

7.1.2 Marshalling Modes

A consist may have different dynamic operation modes. According to these consist operation modes, PDM offers marshalling modes. Each marshalling mode may have a different configuration.

EXAMPLE

A traction consist like a locomotive may be the traction leader of a train set, traction follower or support no traction at all. In each case different data will be imported and exported.

A default mode should always be present, if no specific mode is used for PDM.

If the consist operation mode changes, PDM can be reconfigured to accept the new marshalling mode.

7.1.3 Data Paths in PDM

This subclause is for informational purposes only and not normative as each gateway supplier may be implemented it differently.

PDM marshalls the process data from a source to a destination. The destination is always a Traffic Store. The source is a Traffic Store, a Default Value Buffer or an Undefined Value Buffer.

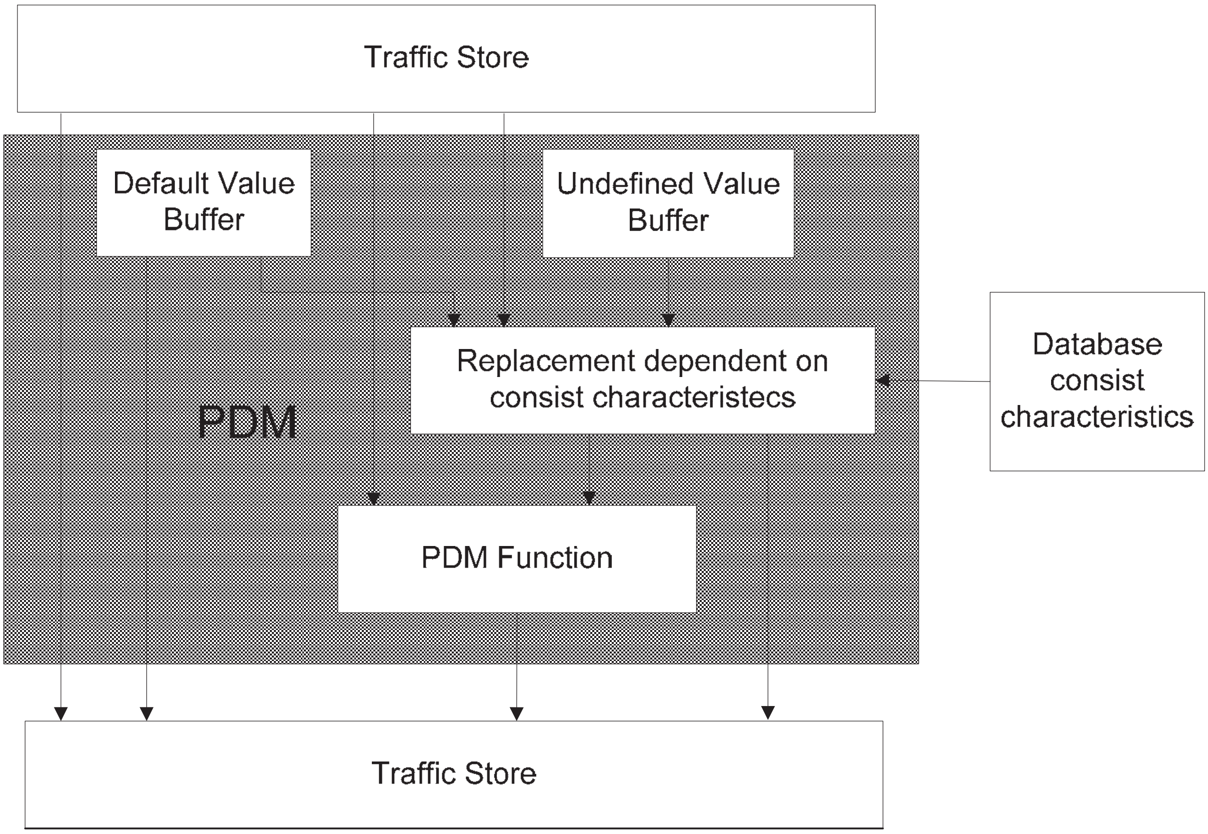


Figure 143 – PDM Data Paths

The following data paths can be configured:

from Traffic Store to Traffic Store, direct or via a PDM Function. This is the basic task of PDM. The process variable can be processed by a function before it is written to the destination Traffic Store.

from Default Value Buffer to Traffic Store Default values are used when a value should be written to a port, but there is in any case no process variable which can deliver this value. Default values are not intended to substitute invalid or too old (not fresh) process variables.

Consist feature dependent process variable from Traffic Store to Traffic Store, direct or via a PDM Function. Process variables may be combined with static and dynamic consist features. Marshalling will only be done, when the consist feature is present. Otherwise PDM substitutes the process variable by an undefined or application defined default value of the apropriate data type. An undefined value is represented by a check variable set to 11_b or the variable value set to all "1". This conforms to the TCN standard IEC 61375.

Three cases should be considered for consist dependent process variables:

One variable with one check variable: PDM can substitute an undefined value by setting the check variable to 11_b.

Several variables with one check variable: If all variables are not supported, PDM can substitute undefined values by setting the check variable to 11_b.

Several variables with one check variable: If only some variables are not supported, PDM should substitute application defined default values.

The process variable can be processed by a function before it is written to the destination Traffic Store. If the process variable is substituted by an undefined value, it can be ignored as function argument.

7.1.4 PDM Operation

PDM is activated by a configurable timer or an event (e.g. WTB data receive). After activation, PDM copies all variables configured for the activated marshalling type.

The copy process has four steps:

a) All variables are read, one data set after the other.

b) For each variable with configured freshness supervision the freshness is checked. Too old variables are invalidated (see below).

c) Then all configured functions (see 7.1.5 PDM Functions) are applied to the variables.

d) At last PDM copies the variables, function results and default values to the destination ports and Traffic Stores.

for all Data-Sets from which Variables come

- read all variables of Data-Set
- *if* (Import Marshalling *and* Frame Type Field ok) *or not* Import Marshalling
  - *then*
    - *for all* variables of Data-Set
      - *if* check freshness, data too old
        - *then*
          - set variable invalid
        - *else*
  - *else*
    - set all variables of Data-Set invalid

if Import Marshalling *and* any frame Type Field not ok

- *then*
  - set all variables of WTB Traffic Store invalid
- *else*

for all PDM Functions

- *execute* PDM Function

for all Data-Sets to which Variables go to

- write all variables of Data-Set
  (PD Variables, Function results, Default Values)

Figure 144 – PDM Operation

A variable or a function result (see chapter 7.1.5) is invalidated by the following algorithm:

if variable or function result has check variable	
then set check to 00_b	*else* set variable value to all "0"

Figure 145 – PDM Invalidate Variable or Function result

If the variable or function result has a check variable, only the check variable is set to 00_b. The variable value is not set to all “0", since the variable cannot be very invalid.

If the variable or function result has no check variable the value will be set to all "0". This conforms to this standard.

NOTE Since overwriting the value 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. (see 6.2.2.2.3).

7.1.5 PDM Functions

7.1.5.1 General

Additionally to the pure copying of variables, PDM supports the processing of process variables by functions. The function processing is supported for all Marshalling types.

EXAMPLE Suppose, an application needs to know whether all doors of a train are closed. Since a train may consist of 1 to 20 consists with doors, the application must be able to process a wide range of input data. Using a function of the PDM which reads all door states and delivers one variable saying "all doors closed" makes application programming much easier.

The functions offered by PDM have generally the form

$y = f(x_1, x_2, \ldots x_n)$; x_i , $i = 1 .. n$, input argument, y function result

There may be any number (greater than zero) of arguments and one result. Arguments of a function may come from different ports and Traffic Stores. Arguments and the result are described by PV_Names.

The PDM offers the following standard processing functions:

- logical functions:
  AND, AND_IGNORE_INVALID
  OR, OR_IGNORE_INVALID
  XOR, XOR_IGNORE_INVALID

- numeric functions:
  MIN, MIN_IGNORE_INVALID
  MAX, MAX_IGNORE_INVALID
  SUM, SUM_IGNORE_INVALID

7.1.5.2 Function processing

All arguments of a function are checked for validity. Too old variables are already invalidated by the second step of the copy process. Arguments can have a check variable or not. Both variants can be mixed if the function has more than one argument.

If an argument is invalid or undefined, it can be ignored (functions XXX_IGNORE_INVALID). If an invalid or undefined argument is not ignored, it sets the function result to invalid.

Only valid arguments are processed. If all arguments are invalid or undefined, the result is set to invalid.

If necessary, the type of the arguments is converted to a type suitable for processing. The processing type is configurable.

If an error occurs during the evaluation of the function the result is set to invalid

After all arguments are processed, the computed result is converted to the desired function result described by a PV_Name.

The function result can have a check variable or not, independent of the arguments.

<table>
<tr><td colspan="4">for all arguments of function</td></tr>
<tr><td rowspan="3"></td><td colspan="3">if variable is valid</td></tr>
<tr><td>then
cast arguments type to computing type</td><td colspan="2">else
if IGNORE_INVALID</td></tr>
<tr><td>apply function to argument and compute function result</td><td>then
ignore argument</td><td>else
set function result invalid
return function result</td></tr>
</table>
<table>
<tr><td colspan="2">if all variables are invalid</td></tr>
<tr><td>then
set function result invalid</td><td rowspan="2">else</td></tr>
<tr><td>return function result</td></tr>
<tr><td colspan="2">cast computing type to result type</td></tr>
</table>
<table>
<tr><td colspan="2">if function result has check</td></tr>
<tr><td>then
set check to 01_b</td><td>else</td></tr>
<tr><td colspan="2">return function result</td></tr>
</table>

Figure 146 – PDM Operation

The validity of a variable is checked by the following algorithm:

<table>
<tr><td colspan="4">if Variable has check</td></tr>
<tr><td colspan="2">then</td><td colspan="2">else</td></tr>
<tr><td colspan="2">if check = 10_b or check = 01_b</td><td colspan="2">if Variable value has all "0" or "1"</td></tr>
<tr><td>then
Variable is valid</td><td>else
Variable is invalid</td><td>then
Variable is invalid</td><td>else
Variable is valid</td></tr>
</table>

Figure 147 – PDM Validty check

7.1.5.3 Logical functions: AND, OR and XOR

For these functions the arguments are of type BOOLEAN, ANTIVALENT or of type BITSET. If the argument is of type BITSET, it is necessary to specify also the bit position of the bit within the bit-set. The types of arguments may be mixed within one function call.

The result value of this functions is of type BOOLEAN or ANTIVALENT. Additionally, it is possible to specify whether the variable is used directly or whether the variable is negated before use.

7.1.5.4 Numeric functions: MIN, MAX, SUM

For these functions the arguments are of type INTEGER, UNSIGNED, REAL and FRACTIONAL with maximal size of 32 bits. INTEGER and UNSIGNED with different sizes may be mixed within one function call. After processing the type of the result is converted (by a cast) to the type of the destination variable. There is no range checking, be careful with overflows.

7.2 WTB Line Fault Location Detection

A diagnosis computer can monitor the WTB line disturbance bits in the gateway node status word. If the diagnosis computer detects a steady disturbance on a line, it can request a Line

Fault Location Detection (LFLD) at the TCN Network Management (TNM). The TNM cooperates with the WTB link layer link management to detect the location of the fault. After the LFLD process has completed, the diagnosis computer can read the result at TNM.

A fault on a line disturbing the proper termination of the line may cause signal reflection, which can affect the communication of several nodes on this line. So the LFLD process opens the interruption relays of the disturbed line at the intermediate nodes. This will produce terminated segments of the WTB, which can be checked.

7.2.1 Architecture

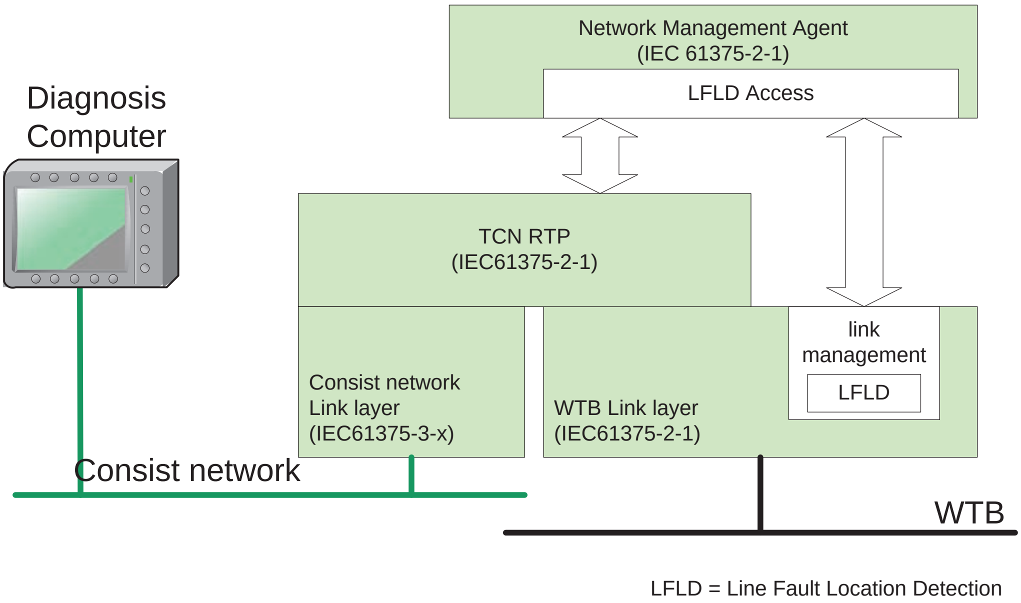


Figure 148 – LFLD Architecture

The WTB Line Fault Location Detection (LFLD) is a function of WTB link layer, which is implemented in the link management. The link management uses TNM to communicate with the other WTB nodes during the LFLD process.

The TCN Network Management offers a new service LFLD in the WTB link services (IEC 61375-1, 8.4.3.6) and supports sub-commands for the diagnosis computer and for the WTB link management

The diagnosis computer can access the TNM service LFLD by TCN message data via MVB to

Start LFLD on an WTB end node

Get LFLD result

Cancel LFLD

The WTB link manager accesses the TNM service LFLD by TCN message data via WTB to

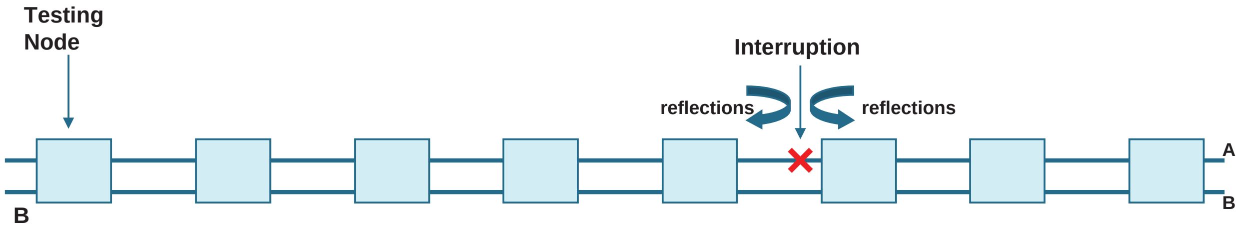
Start and Stop the LFLD process on the other WTB end node

Start and Stop the LFLD process on the WTB intermediate nodes

7.2.2 Protocol Overview

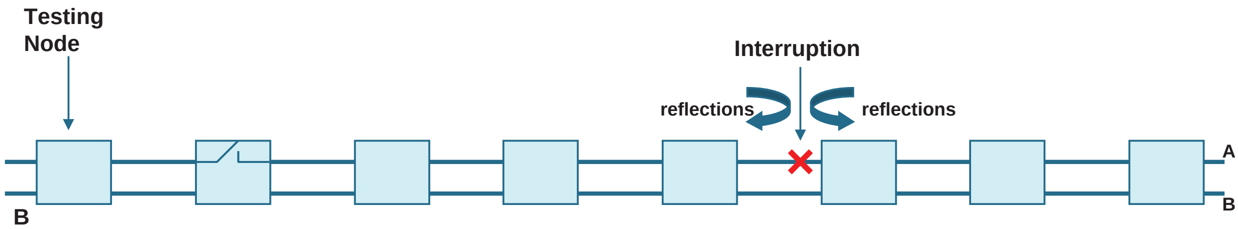
Step1:

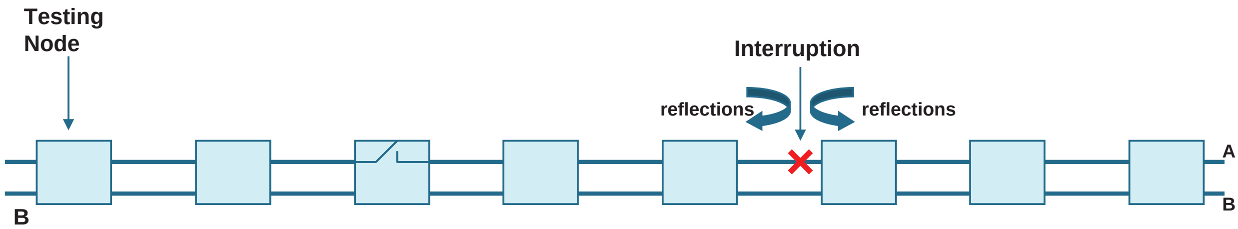
The diagnosis computer requests a LFLD at an end node and makes it a Testing Node (TN). The WTB link management of the TN starts the detection process

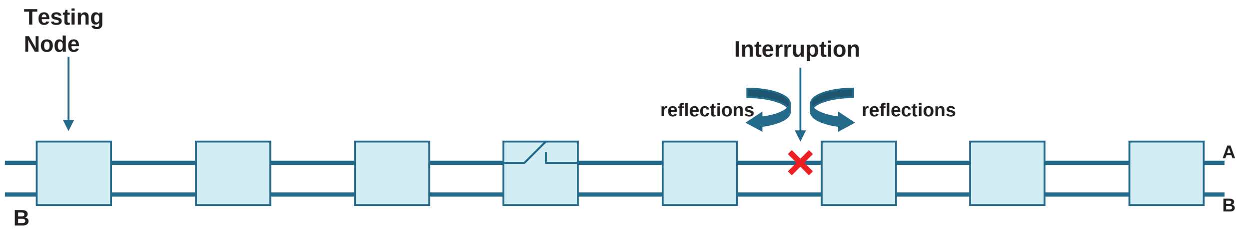


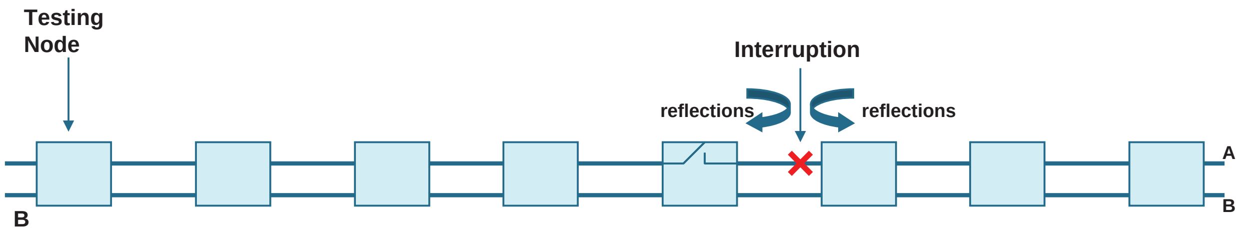
Step2:

The TN asks the next node (= Segmenting Node SN) to open the line switch. If the TN registers no line fault for frames received from the SN for a time T_2, this line segment is assumed to work correctly. Then the testing node asks the node to close the line switch again and continues with the next but one node. This process continues as long as the TN does not register line faults for frames coming from the SN.



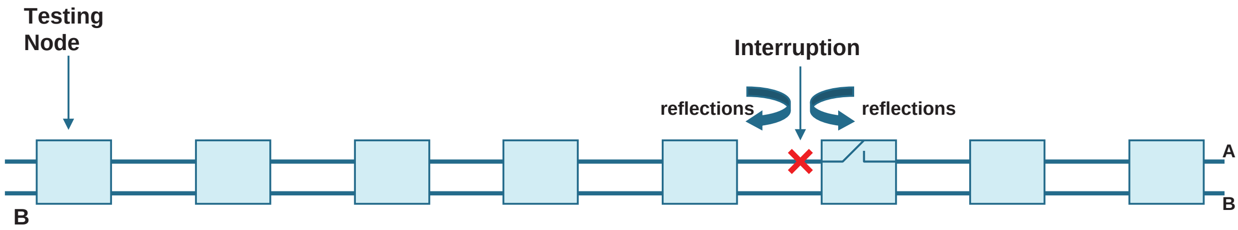






Step3:

Now, the TN not any longer receives without line fault from the SN. This is the indication that there must be some line fault in the last tested segment.



7.2.3 LFLD Sequence

Figure 149 show the LFLD process sequence.

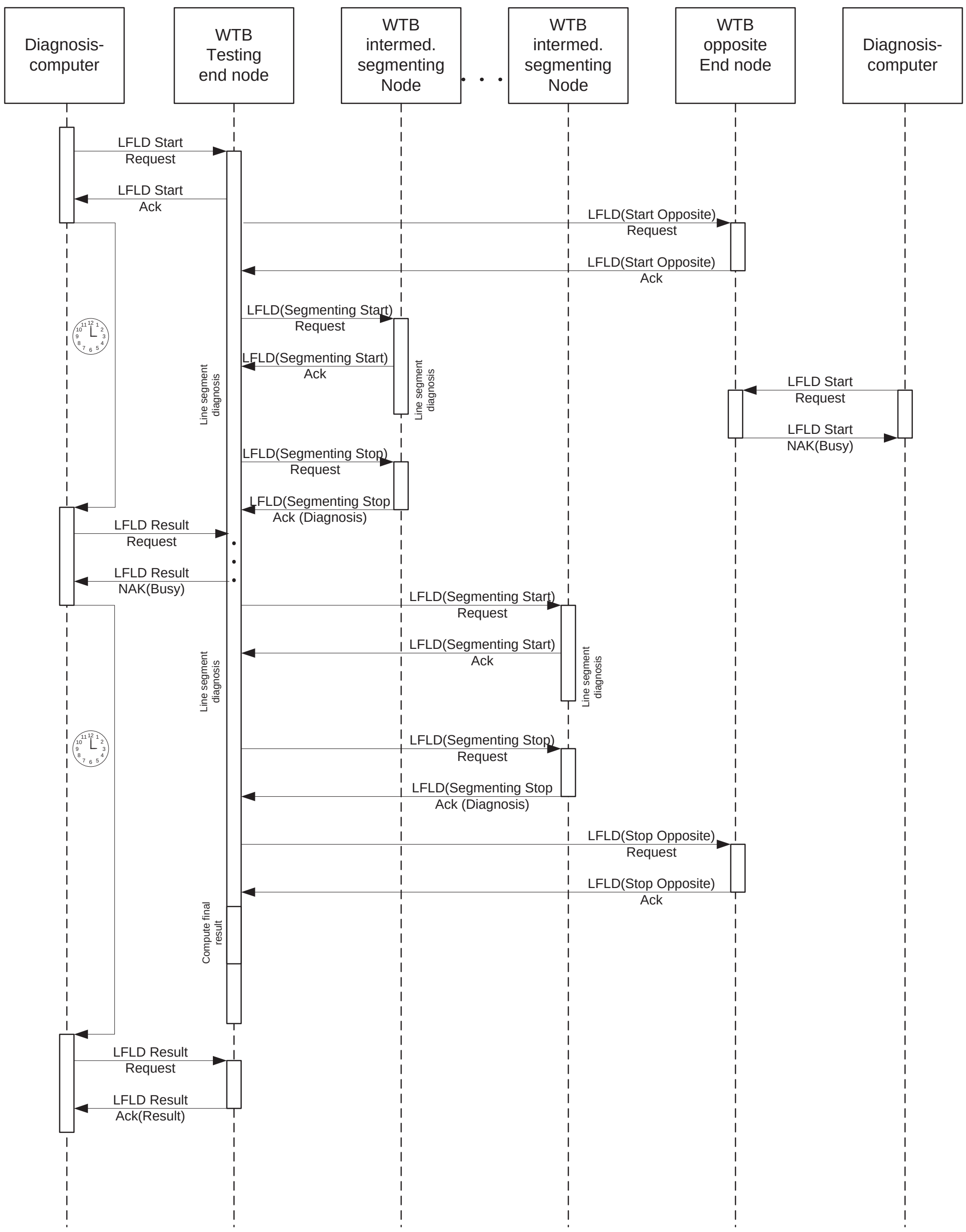


Figure 149 – LFLD sequence

The diagnosis computer starts the LFLD process by a TNM message.

The Testing Node (TN) informs the opposite end node (OE) by a TNM message, that the LFLD process is started. This shall prevent the opposite end node to start also a LFLD process.

When the TN receives the reply message from the opposite end node, it selects the first Segmenting Node (SN).

The TN requests the SN to segment the bus (open interruption relay) and start the line diagnosis.

When the TN receives the reply message from the SN, it supervises the segment from node TN to node SN, if it can receive a Process Data (PD) response frame from the SN or the previous but one node without line disturbance.

The SN supervises the segment from node SN to node OE to receive a PD response frame from node OE without disturbance.

After the longest possible individual period (128 * 25 msec) the TN requests the SN to stop segmenting the bus (close interruption relay) and to report, if the SN could receive node OE without disturbance.

The TN selects the next SN and repeats steps 4 to 7 until the location of the fault is detected or node OE is reached.

When node OE is reached, the TN informs the OE, that the LFLD process has completed.

The diagnosis computer can get the LFLD result from the TN. If the LFLD process is still in progress, the TN refuses the request with an error message. The diagnosis computer has to poll the TN until the result is available.

If a diagnosis computer requests a LFLD at the node OE during the LFLD process, the OE refuses the request with an error message.

7.2.4 End Node State Machine (Testing Node)

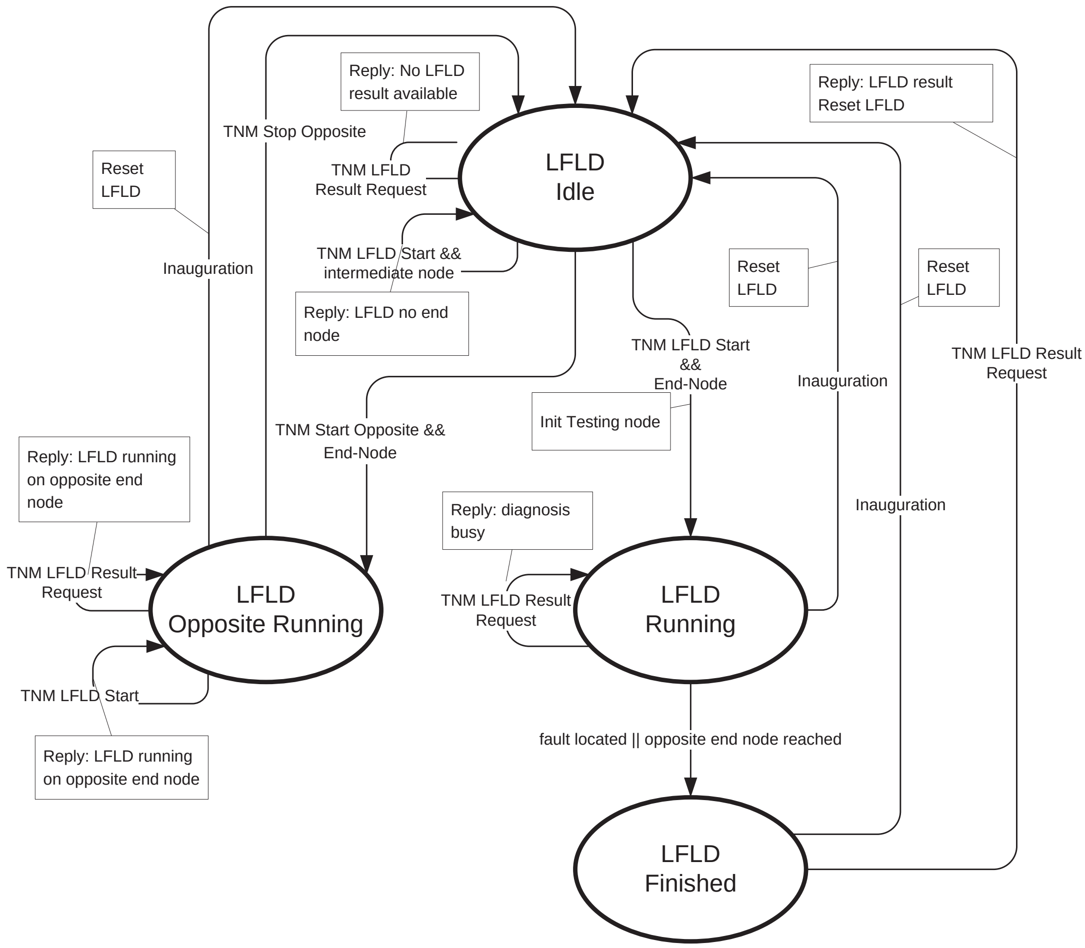


Figure 150 – End node state machine

7.2.5 Intermediate Node State Machine (Segmenting Node)

The implementation of the segmenting node is stateless.

7.2.6 Disturbed Line selection

The disturbed line A or B depends on the orientation of each node.

The TN gets its local disturbed line. The TN deduces the disturbed line of the WTB master depending on the own orientation to the WTB master an uses it as the normalized disturbed line. When the TN starts and stops a SN, the TN requests to open and close the normalized disturbed line. The SN itself deduces from the normalized disturbed line and its orientation to the master, which local line relay to open and close.

7.2.7 Location Detection

The TN initializes this node pair to TN and the direct neighbour of TN. The LFLD process steps from node to node.

EXAMPLE The LFLD process has reached SN 63 and the fault is between node 63 and 1.

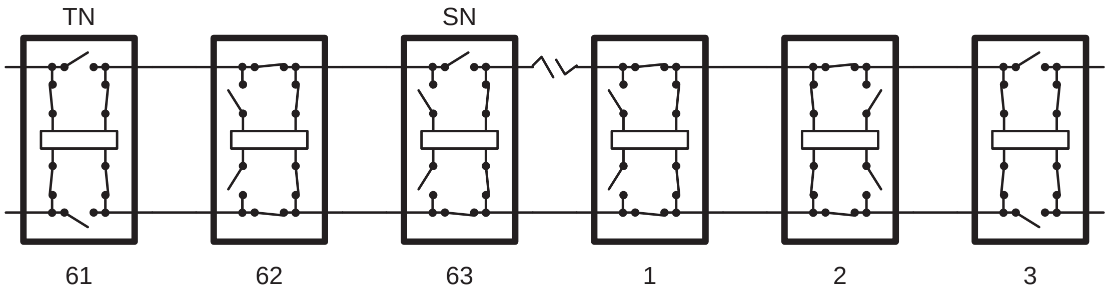


Figure 151 – LFLD process, SN at node 63

In Figure 151 SN 63 cannot be reached by the TN, because the attachment relay is closed on the far side. The TN can see only a good response from the previous node. So the TN cannot distinguish between fault between node 62 and 63 or between node 63 and 1.

So, the SN also monitors the WTB segment on the other side of the interruption relay, in this example from node 63 to 3. The SN reports the quality of the segment to the TN, when the TN requests to close the interruption relay. In this example node 63 detects the segment to be bad and the TN can assume the segment from node 61 to 63 is good, even if the TN cannot get a response from node 63.

So the TN continues with the next SN (node 1).

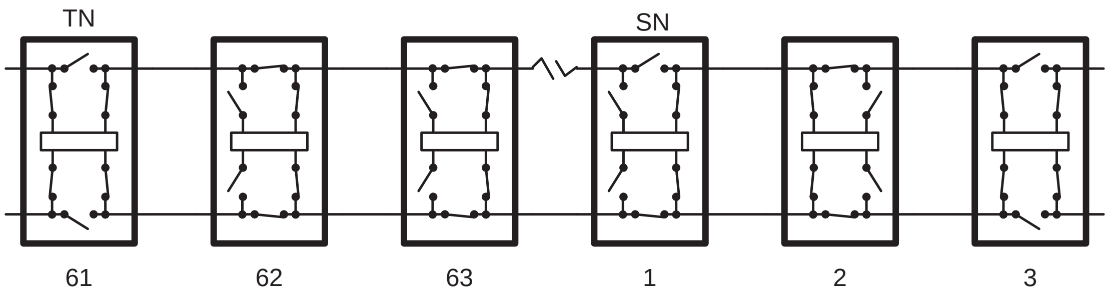


Figure 152 – LFLD process, SN at node 1

The SN (now node 1) monitors segment from node 1 to 3 and detects it as good and reports this to the TN. The TN can deduce, that the fault shall be just before the SN, i.e. between nodes 63 and 1.

When the TN gets the report from the SN it decides:

if the SN reports, that it received node OE without disturbance, to stop the LFLD process and the pair of nodes consists of the SN and the node just before the SN.

if the TN receives the SN or the node just before without disturbance, to set the pair of nodes to the received node and the next node

This decision is done after each stop of a SN.

When the TN has stopped or reached the opposite end node, it has to check, if it has to solve the following case:

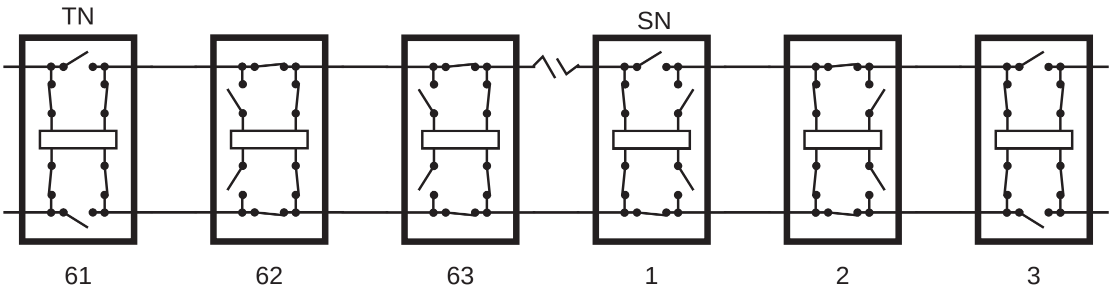


Figure 153 – LFLD process, SN at node 1, attachment in direction 1

The TN may reach neither node 63 nor node 1 and the SN cannot reach node 3. But the TN can deduce that the fault has to be between node 63 and 1, because if it were not, the TN should have detected the segment from node 61 to 1 as good.

The TN decides:

if node pair is (63, 1) and TN is at bottom address and attachment relay of WTB master in direction 2, then correct node pair to (1, 2)

if node pair is (1, 2) and TN is at top address and attachment relay of WTB master in direction 1, then correct node pair to (63, 1)

The final result of the LFLD process is a pair of nodes which delimit the fault location.The result is stored in Train Network Management for retrieval by the diagnosis computer.

8 Train Network Management

8.1 General

8.1.1 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 Consist network 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 CODE 557 or consist identification in accordance with UIC CODE 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.

8.1.2 Structure of this clause

This Clause is structured in the following way:

Subclause 8.1 General

Subclause 8.2 Manager, Agents and interfaces

Subclause 8.3 Managed objects

For each object, the following is specified:

- object description;
- access;
- services offered by the object.

Subclause 8.4 Services and management messages

For each service, the following is specified:

- its description;
- Call_Message and parameters;
- Reply_Message and parameters.

Subclause 8.5 Interface Procedures

For each interface, the procedures to access and control the Agent locally are specified.

8.2 Manager, Agents and interfaces

8.2.1 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.

8.2.2 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 154.

The Manager shall act as a Caller and the Agent as a Replier.

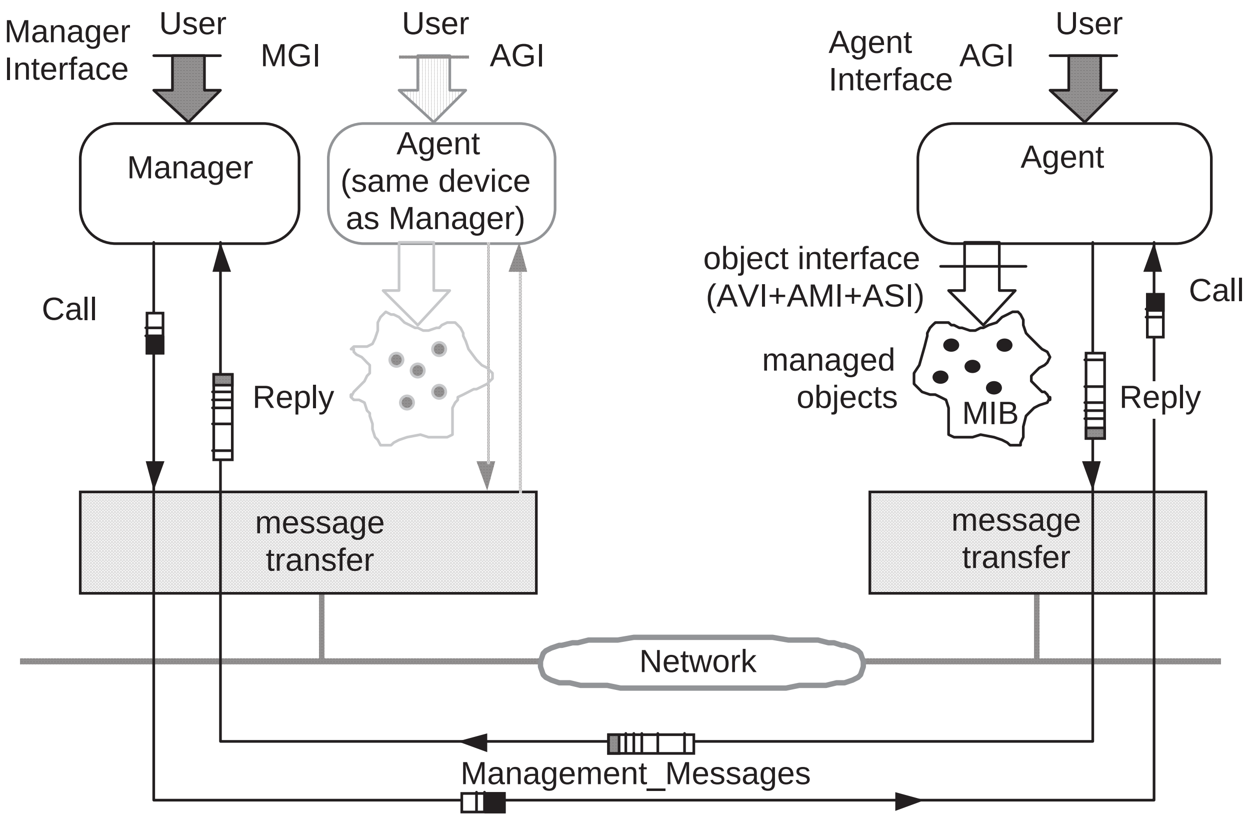


Figure 154 – Management messages

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.

8.2.3 Interfaces

8.2.3.1 Object interface

Communication objects are related to the network communication, while non-communication objects are related to other properties of a Station.

EXAMPLE 1 The configuration of a Bus Administrator is a communication object.

EXAMPLE 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 155). The definition of this interface is included in the clauses which specify the corresponding layer: Clause 6 (RTP), IEC 61375-3-1 (MVB) and this document (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.

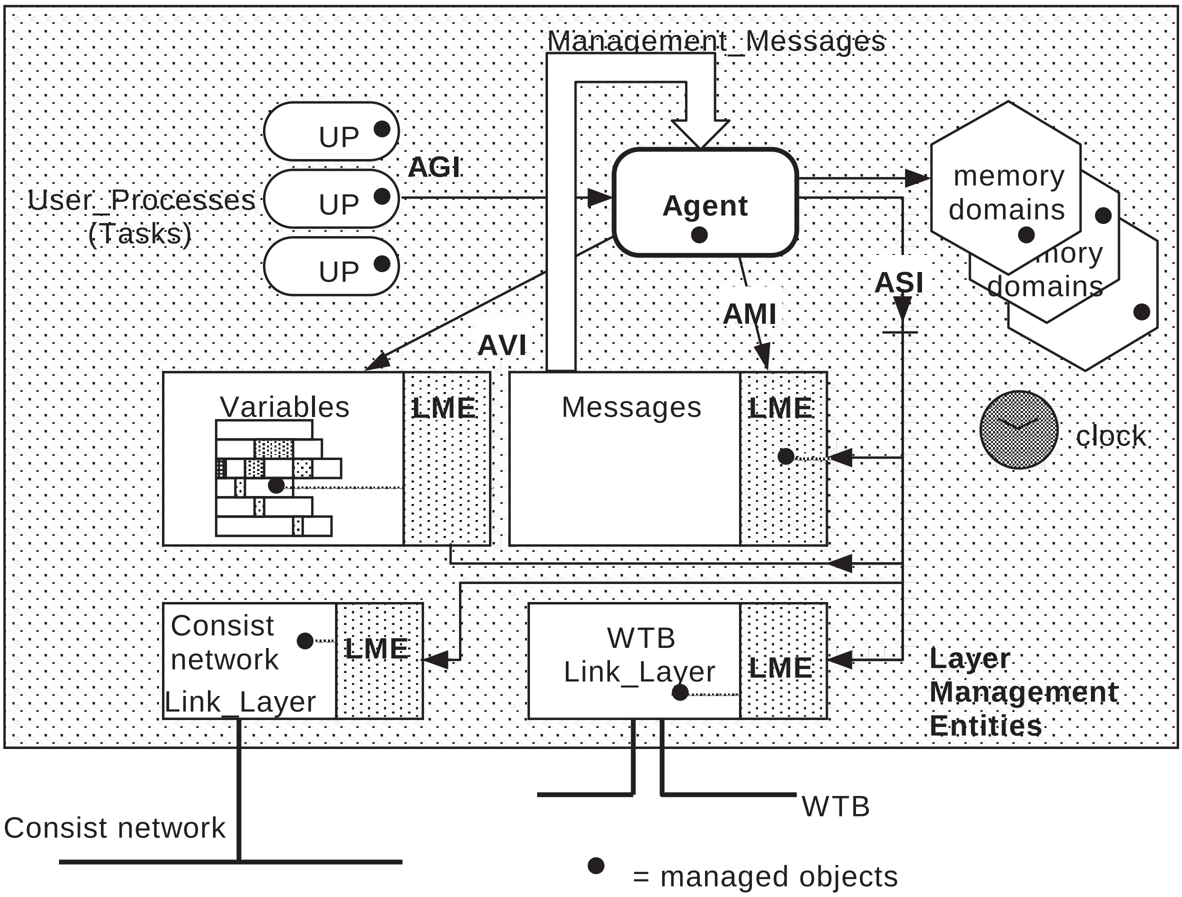


Figure 155 – Agent Interface on a (gateway) Station

8.2.3.2 Manager interface (MGI)

The Manager shall offer a set of procedures to access remote objects, which form the Manager Interface or MGI (see Figure 154).

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.

8.2.3.3 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 8.5.2.

NOTE There are no network messages associated with the Agent interface.

8.3 Managed objects

8.3.1 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.

8.3.2 Station objects

8.3.2.1 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 Consist network attached and no Train Bus, the Station_Status shall be a copy of the Consist network 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 156:

```
Station_Status::=       RECORD
  {
  station_capabilities  BITSET4
    {                                       -- basic capabilities of a
                                               station
    sp    (0),                              -- special device
    ba    (1),                              -- MVB Bus_Administrator
    gw    (2),                              -- gateway or Train Bus node
    md    (3)                               -- messages capability
    },
  class_specific        WORD4,              -- always 0
  common_flags          BITSET8
    {
    lat    (0),                             -- unused, = 0
    lnd    (1),                             -- Link Disturbed
    ssd    (2),                             -- Some System Disturbance
                                               fault in the controlled
                                               process (e.g. power loss)
    sdd    (3),                             -- Some Disturbance of the
                                               Device:
                                               device malfunctions (e.g.
                                               error in checksum)
    scd    (4),                             -- 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.
    frc    (5),                             -- forced Station: a port of
                                               this device is forced
    snr    (6),                             -- Station Not Ready:e.g.
                                               Station not initialised
    ser    (7)                              -- 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 156 – Station_Status

8.3.2.2 Station_Control object

Every Station shall implement a Station_Control object with two associated services:

a) start a Station and assign configuration parameters, such as the Station's name;

b) 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.

8.3.2.3 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_layer15  (15),   --                  first transmitted bit
  link_layer14  (14),
  link_layer13  (13),
  link_layer12  (12),
  link_layer11  (11),
  link_layer10  (10),
  link_layer9    (9),
  link_layer8    (8),
  link_layer7    (7),
  link_layer6    (6),
  link_layer5    (5),
  link_layer4    (4),
  link_layer3    (3),
  link_layer2    (2),
  link_layer1    (1),
  link_layer0    (0)
  }
```

8.3.2.4 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.

8.3.3 WTB link objects

8.3.3.1 WTB Status object

Each node attached to the Train Bus shall implement a read-only WTB_Status object, which is specified in 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 this standard:

a) Node_Descriptor, a structure defining the Process_Data frame, (see 5.5.2.1);
b) Node_Report, a BITSET8 which signals disturbances, (see 5.5.2.2);
c) User_Report, a WORD8 supplied by the application (see 5.5.2.3);
d) Node_Key, a RECORD containing the Node_Type and Node_Version (see 5.5.2.4);
e) Ls_T_State, an ENUM8 which indicates in which states a node is (see 5.6.4.2.2);
f) Node_Strength, an ENUM8 expressing the strength of the node (see 5.6.4.16.3).

8.3.3.2 WTB Topography object

Each node attached to the Train Bus shall implement a read-only WTB_Topography object, which is specified in this standard ("Wire Train Bus").

The Agent accesses this object through the link management interface, ls_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 6.3.

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.

8.3.4 Variable objects

8.3.4.1 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 (Consist network) 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::=             RECORD
  {
  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)
  }
```

8.3.4.2 Variables object

8.3.4.2.1 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 Consist network 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.

8.3.4.2.2 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.

8.3.4.2.3 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 Consist network 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.

8.3.4.3 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.

8.3.5 Messenger objects

8.3.5.1 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.

8.3.5.2 Messenger control

Each Station providing the Messages service shall implement the Messenger control object, which allows the remote configuration of Message communication parameters.

8.3.5.3 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 to which groups it belongs.

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.

8.3.6 Domain objects

8.3.6.1 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, a domain 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.

8.3.6.2 Services

The services on Domain objects shall comprise:

a) setup 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.

8.3.7 Task objects

8.3.7.1 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.

8.3.7.2 Access

The Agent is assumed to access the scheduler to start and stop tasks.

The access to tasks is implementation-dependent.

8.3.7.3 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 8.2.3.3) except when "override" access is specified.

8.3.8 Clock object

8.3.8.1 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.

8.3.8.2 Services

Two clock services are specified:

a) read_clock, which reads the current time value;
b) set_clock, which sets the clock value.

8.3.9 Journal object

8.3.9.1 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.

8.3.9.2 Services

One journal service is specified:

– read_journal, which reads the last entries of the journal.

8.3.10 Equipment object

8.3.10.1 Object

A Station may implement an Equipment Descriptor which describes the supported equipment.

8.3.10.2 Services

One service is specified:

– read_equipment_descriptor, which retrieves a pointer to the memory location where the equipment descriptor is located

8.4 Services and management messages

8.4.1 Notation for all management messages

8.4.1.1 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.

8.4.1.2 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_WTB_USER_REPORTport (25),
  LINE_FAULT_LOCATION_DETECTION (26),
  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.
  }
```

NOTE SIF_codes for MVB services are included for completeness. For a description of the MVB services see IEC 61375-3-1.

8.4.1.3 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,
}
  }
```

8.4.1.4 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]            Replay_Read_Equipment
}
  }
```

8.4.1.5 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
  }
```

8.4.1.6 Bit numbering and data encoding

The notion of the management messages is structured into 16-bit units. One 16-bit unit can hold two 8-bit types, one 16-bit type or a part of a 32-bit type. The following mappings show the bit numbering of the bits depending on the types.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
UNSIGNED8, BITSET8, ENUM8, WORD8								UNSIGNED8, BITSET8, ENUM8, WORD8							
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
UNSIGNED16, ENUM16, WORD16															
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

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
BITSET16															
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
UNSIGNED32															
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

The byte with the lower bit numbering shall be transmitted first.

8.4.2 Station services

8.4.2.1 Read_Station_Status

8.4.2.1.1 Description

This service reads the Station_Status object remotely.

8.4.2.1.2 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
```

8.4.2.1.3 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.

8.4.2.2 Write_Station_Control

8.4.2.2.1 Description

This service resets a Station and assigns its Station name and Station_Identifier.

8.4.2.2.2 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.
  }
```

8.4.2.2.3 Reply_Write_Station_Control

0–7	8–15
tnm_key	sif_code = 1
bus_id	reserved1
device_address	

```
Reply_Write_Station_Control::= RECORD      -- sif_code = 1
  {
  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
  }
```

8.4.2.3 Read_Station_Inventory

8.4.2.3.1 Description

This service reads the inventory object.

8.4.2.3.2 Call_Read_Station_Inventory

0–7	8–15
tnm_key	sif_code = 2

```
Call_Read_Station_Inventory::= RECORD
  {}                     --                no parameters
```

8.4.2.3.3 Reply_Read_Station_Inventory

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="8">tnm_key</td><td colspan="8">sif_code = 2</td></tr>
<tr><td colspan="16">reserved1</td></tr>
<tr><td colspan="16">agent_version: STRING32</td></tr>
<tr><td colspan="8">(CHARACTER8)</td><td colspan="8">CHARACTER8 or ‘00’H</td></tr>
<tr><td colspan="16">manufacturer_name: STRING32</td></tr>
<tr><td colspan="8">(CHARACTER8)</td><td colspan="8">CHARACTER8 or ‘00’H</td></tr>
<tr><td colspan="16">device_type: STRING32</td></tr>
<tr><td colspan="8">(CHARACTER8)</td><td colspan="8">CHARACTER8 or ‘00’H</td></tr>
<tr><td colspan="16">service_set: BITSET256</td></tr>
<tr><td>SV0</td><td colspan="14"></td><td>SV255</td></tr>
<tr><td>LL0</td><td colspan="14">link_set</td><td>LL15</td></tr>
<tr><td colspan="8">reserved2</td><td colspan="8">station_id</td></tr>
<tr><td colspan="16">station_name: STRING32</td></tr>
<tr><td colspan="8">(CHARACTER8)</td><td colspan="8">CHARACTER8 or ‘00’H</td></tr>
<tr><td colspan="16">station_status</td></tr>
</table>

```
Reply_Read_Station_Inventory::= RECORD
  {
  reserved1            WORD16 (=0)        -- used for alignment
  agent_version        STRING32           -- version of the Agent
                                             (e.g. reference to this
                                             Clause).
  manufacturer_name    STRING32           -- name of manufacturer of the
                                             device
  device_type          STRING32           -- name of device and serial
                                             number.
  service_set          BITSET256          -- one bit for each of the
                                             services.
                                             (the first bit set to 1
                                             means the device supports
                                             service
                                             Read_Station_Status)
  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.

8.4.2.4 Write_Station_Reservation

8.4.2.4.1 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 8.5.2.4).

8.4.2.4.2 Call_Write_Station_Reservation

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 3							
command															
access_type															
reservation_time_out															
manager_id															

```
Call_Write_Reservation::= RECORD
  {
  command          ENUM16
    {
    RESERVE,       (1),                    -- reserve the device for this
                                              manager
    KEEPREL        (2),                    -- release and keep changes
    STARTREL       (3)                     -- release and restart.
    },
  access_type      ENUM16
    {
    WRITEREQ       (0),                    -- write access requested
    OVERRIDE       (1)                     -- override access reserved.
    },
```

```
  reservation_time_out  UNSIGNED16,         -- time during which the
                                               Station remains reserved, in
                                               multiples of 1 s, but no
                                               larger than 3600 s.
  manager_id       UNSIGNED32               -- identifies the Manager (use
                                               of this identifier is
                                               implementation-dependent).
  }
```

8.4.2.4.3 Reply_Write_Station_Reservation

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 3							
reserved1															
manager_id															

```
Reply_Write_Reservation::= RECORD
  {
  reserved1            WORD16               -- for alignment
  manager_id           UNSIGNED32           -- identifies the Manager (use
                                               of this identifier is
                                               implementation-dependent).
  }
```

8.4.2.5 Read_Service_Descriptor

8.4.2.5.1 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.

8.4.2.5.2 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
  }
```

8.4.2.5.3 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
  }
```

8.4.2.6 Read_Links_Descriptor

8.4.2.6.1 Description

This service reads the Links_Descriptor, the description text which shall describe each link layer present in the Station_Inventory.

8.4.2.6.2 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
```

8.4.2.6.3 Reply_Read_Links_Descriptor

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 6							
nr_links															
links_descriptor: ARRAY [nr_links] OF															
bus_id								link_type							
link_name: STRING32															
								CHARACTER8 or ‘00’H							

```
Reply_Read_Links_Descriptor::= RECORD
  {
  nr_links              UNSIGNED16 (0..15), -- number of supported links
  links_descriptor      ARRAY [nr_links] OF
    {                                       -- list of existing links,
                                               comprising for each:
    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 as
                                               character string
    }
  }
```

8.4.2.7 Write_Links_Descriptor

8.4.2.7.1 Description

This service writes the Links_Descriptor, the description text which shall describe each link layer present in the Station_Inventory.

8.4.2.7.2 Call_Write_Links_Descriptor

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 7							
nr_links															
links_descriptor: ARRAY [nr_links] OF															
bus_id								link_type							
link_name: STRING32															
								CHARACTER8 or ‘00’H							

```
Call_Write_Links_Descriptor::= RECORD
  {
  nr_links              UNSIGNED16 (0..15), -- number of supported links
  links_descriptor      ARRAY [nr_links] OF
    {                                       -- list of existing links,
                                               comprising for each:
    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
      LINK_CAN       (5),                   -- CAN bus
      LINK_ETH       (6)                    -- Ethernet
                                            -- other reserved
      },
    link_name           STRING32            -- name of the link
    }
  }
```

8.4.2.7.3 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
```

8.4.3 WTB link services

8.4.3.1 Read_Wtb_Status

8.4.3.1.1 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 Type_WTBStatus and Type_LLStatisticData in 5.6.4.16.3 and 5.6.4.22.3.

NOTE In case of discrepancy, the parameter definitions in 5.6.4.16.3 and 5.6.4.22.3 hold.

8.4.3.1.2 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
  }
```

8.4.3.1.3 Reply_Read_Wtb_Status

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="8">tnm_key</td><td colspan="8">sif_code = 20</td></tr>
<tr><td colspan="8">bus_id</td><td colspan="8">node_address</td></tr>
<tr><td colspan="8">wtb_hardware_id</td><td colspan="8">wtb_software_id</td></tr>
<tr><td colspan="8">hardware_state</td><td colspan="8">link_layer_state</td></tr>
<tr><td colspan="8">net_inhibit</td><td colspan="8">node_address</td></tr>
<tr><td colspan="8">node_orient</td><td colspan="8">node_strength</td></tr>
<tr><td colspan="8">node_frame_size</td><td colspan="8">node_period</td></tr>
<tr><td colspan="8">node_type</td><td colspan="8">node_version</td></tr>
<tr><td colspan="8">node_report</td><td colspan="8">user_report</td></tr>
<tr><td colspan="16">basic_period_count</td></tr>
<tr><td colspan="16"></td></tr>
<tr><td colspan="16">inauguration_count</td></tr>
<tr><td colspan="16">topography_count</td></tr>
<tr><td colspan="16">transmitted_md_count</td></tr>
<tr><td colspan="16">received_md_count</td></tr>
<tr><td colspan="16">line_status_a1
(transmitted_count)

(received_count)
(errors_count)
(timeouts_count)</td></tr>
<tr><td colspan="16">line_status_a2</td></tr>
<tr><td colspan="16">line_status_b1</td></tr>
<tr><td colspan="16">line_status_b2</td></tr>
<tr><td colspan="16">line_switch_count</td></tr>
</table>

```
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
}
```

8.4.3.2 Write_Wtb_Control

8.4.3.2.1 Description

Sets parameters in the Train Bus node.

8.4.3.2.2 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	csl	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
    csl        (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
    }
  }
```

8.4.3.2.3 Reply_Write_Wtb_Control

0–7	8–15
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
  }
```

8.4.3.3 Read_Wtb_Nodes

8.4.3.3.1 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).

8.4.3.3.2 Call_Read_Wtb_Nodes

0–7	8–15
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
  }
```

8.4.3.3.3 Reply_Read_Wtb_Nodes

0–7	8–15
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
    }
  }
```

8.4.3.4 Read_Wtb_Topography

8.4.3.4.1 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.

8.4.3.4.2 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
  }
```

8.4.3.4.3 Reply_Read_Wtb_Topography

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="8">tnm_key</td><td colspan="8">sif_code = 24</td></tr>
<tr><td colspan="8">bus_id</td><td colspan="8">node_address</td></tr>
<tr><td colspan="8">node_orient</td><td>rsv1</td><td>rsv2</td><td colspan="6">topo_counter</td></tr>
<tr><td colspan="8">individual_period</td><td colspan="8">is_strong</td></tr>
<tr><td colspan="8">number_of_nodes</td><td colspan="8">bottom_address</td></tr>
<tr><td colspan="8">top_address</td><td colspan="8">reserved1</td></tr>
<tr><td colspan="8">inaug_data_max_size</td><td colspan="8">nr_descriptors</td></tr>
<tr><td colspan="16">node_descriptions ARRAY [nr_descriptors] OF</td></tr>
<tr><td colspan="8">node_type</td><td colspan="8">node_version</td></tr>
<tr><td>sam</td><td>rsv1</td><td colspan="6">node_address</td><td colspan="8">inauguration_data_size</td></tr>
<tr><td colspan="16">inauguration_data ARRAY ALIGNED16 [inauguration_data_size] OF</td></tr>
<tr><td colspan="8">WORD8</td><td colspan="8">WORD8</td></tr>
</table>

```
Reply_Read_Wtb_Topography::= RECORD
  {
  bus_id                 UNSIGNED8 (0..15),  -- identifier of the link
  node_address           UNSIGNED8,          -- Node_Address to which this
                                                Station is connected
  node_orient            ENUM8,              -- orientation of this node
                                                with respect to master:
                                                0 = unknown, 1 = same, 2 =
                                                inverse
  rsv1                   WORD1 (=0)          -- reserved, =0
  rsv2                   WORD1 (=0),         -- reserved, =0
  topo_counter           UNSIGNED6,          -- the six bits of the
                                                topography counter in this
                                                node.
  individual_period      UNSIGNED8,          -- 8-bit integer representing
                                                the individual period
                                                allocated by the master as
                                                power of two of the  basic
                                                period in ms.
  is_strong              ENUM8,              -- is_strong = 1: bus
                                                controlled by a strong
                                                master, is_strong = 0: bus
                                                controlled by a weak master
  number_of_nodes        UNSIGNED8 (0..63),  -- number of nodes in the
                                                composition
  bottom_address         UNSIGNED8,          -- Node_Address of the node
                                                with the lowest address
  top_address            UNSIGNED8,          -- Node_Address of the node
                                                with the highest address
  inaug_data_max_size   UNSIGNED8            -- copy of maxinaugdatasize
  nr_descriptors         UNSIGNED8           -- in error-free conditions,
                                                equal number_of_nodes
```

```
  node_descriptions  ARRAY[nr_descriptors] OF
    {                                     --  list of descriptions,
                                              beginning with bottom node,
                                              in ascending node address,
                                              consisting of:
    node_type           UNSIGNED8,        --  first part of Node_Key
    node_version        UNSIGNED8,        --  second part of Node_Key
    sam                 BOOLEAN1          --  same direction as master
    rsv1                WORD1 (=0)        --  reserved, =0
    node_address        UNSIGNED6         --  node address
    inauguration_data_size  UNSIGNED8     --  size of the following data
    inauguration_data  ARRAY [inauguration_data_size] OF
                        WORD8             --  inauguration data, user-
                                              supplied structure.
    }
  }
```

8.4.3.5 Write_Wtb_User_Report

8.4.3.5.1 Description

Sets parameters in the Train Bus node.

8.4.3.5.2 Call_Write_Wtb_User_Report

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 25							
bus_id								ur7	ur6	ur5	ur4	ur3	ur2	ur1	ur0

```
Call_Write_Wtb_User_Report::= RECORD
  {
  bus_id               UNSIGNED8 (0..15),    -- identifier of the link
  command              BITSET8
    {
    ur7       (0),                           -- user report bit7
    ur6       (1),                           -- user report bit6
    ur5       (2),                           -- user report bit5
    ur4       (3),                           -- user report bit4
    ur3       (4),                           -- user report bit3
    ur2       (5),                           -- user report bit2
    ur1       (6),                           -- user report bit1
    ur0       (7),                           -- user report bit0
    }
  }
```

8.4.3.5.3 Reply_Write_Wtb_User_Report

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 25							
bus_id								reserved1							

```
Reply_Write_Wtb_User_Report::= RECORD
  {
  bus_id                 UNSIGNED8 (0..15),  -- identifier of the link
  reserved1              WORD8 (=0)          -- reserved
  }
```

8.4.3.6 Line_Fault_Location_Detection LFLD

8.4.3.6.1 Description

The service locates the failure of a disturbed line to be between two nodes. If this Station has no Train Bus, it returns an error and the Reply_Message consists only of the header.

The service has several sub-commands. The following sub-commands are provided for a network management application:

- Start LFLD
- Get LFLD result
- Cancel LFLD

The following sub-commands are provided for internal use.

- Indicate LFLD start to opposite end node
- Indicate LFLD stop to opposite end node
- Start LFLD segmenting node (intermediate node)
- Stop LFLD segmenting node (intermediate node)

To use the service Line_Fault_Location_Detection a multi-stage process should be applied. The diagnosis application should monitor the line status bits to detect a line disturbance. If a steady line disturbance is detected, the diagnosis application should send the Line_Fault_Location_Detection sub-command 0 to an end node, to start the LFLD process. Then the diagnosis application has to poll for the LFLD result with the sub-command 1 until the result indicates the availability of the result. The LFLD result can be got only once. To get a new LFLD result, the LFLD process has to be started again.

To cancel a running LFLD process, the diagnosis application has to issue the Line_Fault_Location_Detection sub-command 2 to the end node, at which the LFLD process was started with sub-command 0.

NOTE 1 If there are several failures on the disturbed line, only one can be detected.

NOTE 2 If there are nodes, which do not support the service Line_Fault_Location_Detection, the location may only located between several nodes.

NOTE 3 If an inauguration occurs, the LFLD process will be aborted.

8.4.3.6.2 Call_Line_Fault_Location_Detection

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 26							
bus_id								sub_command							
par1								par2							

```
Call_Line_Fault_Location_Detection ::= RECORD
  {
  bus_id                UNSIGNED8 (0..15)   -- identifier of the link
  sub_command           WORD8 (0..6)        -- sub-command
  par1                  WORD8               -- parameter 1 of LFLD
                                               sub_command
  par2                  WORD8               -- parameter 2 of LFLD
                                               sub_command
  }
```

The following Line_Fault_Location_Detection sub-commands are defined:

- Sub-command = 0: Start LFLD process. This sub-command can only be sent to an end node.

par1 = 0, par2 = 0: Not used.

- Sub-command = 1: Get LFLD result. This sub-command can only be sent to an end node.

par1 = 0, par2 = 0: Not used.

- Sub-command = 2: Cancel LFLD process. This sub-command can only be sent to an end node.

par1 = 0, par2 = 0: Not used.

- Sub-command = 3: Indicate LFLD start to opposite end node.

par1 = 0, par2 = 0: Not used.

- Sub-command = 4: Indicate LFLD stop to opposite end node.

par1 = 0, par2 = 0: Not used.

- Sub-command = 5: Start LFLD segmenting node (intermediate node).

par1 = *line* Normalized disturbed line as seen by the WTB bus master (*line* = 0 means line A, *line* = 1 means line B)

par2 = *oe* node address *oe* of the opposite end node

- Sub-command = 6: Stop LFLD segmenting node (intermediate node).

par1 = *line* Normalized disturbed line as seen by the WTB bus master (*line* = 0 means line A, *line* = 1 means line B)

par2 = 0 not used

8.4.3.6.3 Reply_Line_Fault_Location_Detection

0–7	8–15
tnm_key	sif_code = 26
bus_id	sub-command
result	ret1
ret2	reserved

```
Reply_Line_Fault_Location_Detection ::= RECORD
  {
  bus_id             UNSIGNED8 (0..15)   -- identifier of the link
  sub_command        WORD8 (0..6)        -- sub-command
  result             WORD8               -- execution result
  ret1               WORD8               -- return value 1 of LFLD
                                            sub_command
  ret2               WORD8               -- return value 2 of LFLD
                                            sub_command
  reserved           WORD8 (=0)          -- reserved
}
```

The following replies to Line_Fault_Location_Detection sub-command are defined:

- Sub-command = 0 Start LFLD process.

result = 0 LFLD process is started

= 1 LFLD process already running

= 4 command sent to intermediate node: LFLD process not started

= 5 no line disturbed: LFLD process not started

= 6 both lines (temporary) disturbed: LFLD process cannot start

ret1, ret2 = 0 not used

- Sub-command = 1 Get LFLD result.

result = 0 LFLD process is not started, no result available

= 1 LFLD process is just running, no result available

= 2 LFLD result is available

ret1 = *n1* one node address *n1* delimiting line fault

ret2 = *n2* other node address *n2* delimiting line fault

- Sub-command = 2 Cancel LFLD process.

result = 0 LFLD process is not started on this end node

= 1 LFLD process cancelled

ret1, ret2 = 0 not used

- Sub-command = 3: Indicate LFLD start to opposite end node.

result = 0: LFLD process is started at opposite end node

ret1, ret2 = 0 not used.

- Sub-command = 4: Indicate LFLD stop to opposite end node.

result = 0: LFLD process is stopped at opposite end node

ret1, ret2 = 0 not used.

- Sub-command = 5: Start LFLD segmenting node

result = 0: LFLD segmenting node is started

ret1, ret2 = 0 not used

- Sub-command = 6: Stop LFLD segmenting node

result = 0 LFLD segmenting node is stopped

ret1= *oe* node address *oe* of opposite end node, if a frame was successfully received

127, if no frame from opposite end node was successfully received

ret2 = 0 Attachment relay in direction 1 closed

= 1 Attachment relay in direction 2 closed

8.4.4 Variables services

8.4.4.1 Read_Ports_Configuration

8.4.4.1.1 Description

Retrieves the list of configured ports with their F_code (which implies the size) and attributes.

8.4.4.1.2 Call_Read_Ports_Configuration

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 30							
bus_id								reserved1 = 0							

```
Call_Read_Ports_Configuration::= RECORD
  {
  bus_id                UNSIGNED8,          -- (0..15)
  reserved1             WORD8 (=0)          -- reserved
  }
```

8.4.4.1.3 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							

```
Reply_Read_Ports_Configuration::= RECORD
  {
  nr_ports              UNSIGNED16,         -- number of ports in that
                                               Traffic_Store (up to 4096)
  ports_list            ARRAY [nr_ports] OF
    {
    bus_id              WORD4,              -- traffic store
    port_address        UNSIGNED12,         -- address of the port
    f_code              F_Code,             -- F_code of the port
    port_config         BITSET4
      {
      src                                   -- publisher port (source)
      snk                                   -- subscriber port (sink)
      twc                                   -- transfer with checksum
      frc                                   -- forced port
      },
    port_size           UNSIGNED8           -- maximum size of the port in
                                               octets (only for WTB)
    }
  }
```

8.4.4.2 Write_Ports_Configuration

8.4.4.2.1 Description

Enables or disables ports.

8.4.4.2.2 Call_Write_Ports_Configuration

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 31							
nr_ports															
ports_list: ARRAY [nr_ports] OF															
bus_id				port address											
f_code				src	snk	twc	frc	port_size							

```
Call_Write_Ports_Configuration::= RECORD
  {
  nr_ports              UNSIGNED16,          -- number of ports in that
                                                Traffic_Store (up to 4096)
  ports_list            ARRAY [nr_ports] OF
    {
    bus_id              UNSIGNED8 (0..15),   -- traffic store id as seen by
                                                the station
    port_address        UNSIGNED12,          -- address of the port
    f_code              F_Code,              -- F_code (see Port
                                                Configuration object)
    port_config         BITSET4
      {
      src                                    -- enables publisher port
                                                (source)
      snk                                    -- enables subscriber port
                                                (sink)
      twc                                    -- enables transfer with
                                                checksum
      frc                                    -- forces port to default value
      },
    port_size           UNSIGNED8            -- maximum size of the port in
                                                octets (only for WTB)
    }
  }
```

8.4.4.2.3 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
  }
```

8.4.4.3 Read_Variables

8.4.4.3.1 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 WTB. i.e. in big-endian format.

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.

EXAMPLE 1 An 8-bit integer with the value ‘0111 1111’B (+127) is transmitted as ‘0000 0000 0111 1111’B.

EXAMPLE 2 An 8-bit integer with the value ‘1111 1111’B (-1) is transmitted as ‘1111 1111 1111 1111’B.

8.4.4.3.2 Call_Read_Variables

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="8">tnm_key</td><td colspan="8">sif_code = 32</td></tr>
<tr><td colspan="16">nr_vars</td></tr>
<tr><td colspan="16">variables_list: ARRAY [nr_vars] OF</td></tr>
<tr><td colspan="4">bus_id</td><td colspan="12">port_address</td></tr>
<tr><td colspan="8">var_size</td><td colspan="8">var_type</td></tr>
<tr><td colspan="16">var_offset</td></tr>
<tr><td colspan="16">chk_offset</td></tr>
</table>

```
Call_Read_Variables::= RECORD
  {
  nr_vars               UNSIGNED16,         -- number of variables in the
                                               list.
  variables_list        ARRAY [nr_vars] OF
    {                                       -- list with one entry for each
                                               variable.
    bus_id              UNSIGNED4,          -- identifier of traffic store
    port_address        WORD12,             -- port address in the traffic
                                               store
    var_size            UNSIGNED8,          -- (simple types:  size in bits
                                               or structured types:  number
                                               of elements according to 6.2
                                               (RTP)
```

```
    var_type          UNSIGNED8,          -- type of the variable
                                             according to 6.2 (RTP)
    var_offset        UNSIGNED16,         -- offset of the variable
    chk_offset        UNSIGNED16          -- offset of the check variable
                                             ('FFFF'H if unused).
    }
  }
```

8.4.4.3.3 Reply_Read_Variables

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 32							
nr_vars															
values_list: ARRAY [nr_vars] OF:															
value (right-justified if smaller than 16 bits)															
check_var		freshness counter													

```
Reply_Read_Variables::= RECORD
  {
  nr_vars             UNSIGNED16,         -- number of variables in the
                                             list.
  values_list         ARRAY [nr_vars] OF
    {                                     -- for each variable in the
                                             list and in same order):
    value             ANY,                -- 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.

8.4.4.4 Write_Force_Variables

8.4.4.4.1 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.

8.4.4.4.2 Call_Write_Force_Variables

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 33							
nr_vars															
variables_list: ARRAY [nr_vars] OF															
bus_id				port_address											
var_size								var_type							
var_offset															
chk_offset															
values_list: ARRAY ALIGN16 [nr_vars] OF															
forced_value															

```
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 6.2
                                               (RTP)
    var_type            UNSIGNED8,          -- type of the variable
                                               according to 6.2 (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.
    }
  }
```

8.4.4.4.3 Reply_Write_Force_Variables

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 33							

```
Reply_Write_Force_Variables::= RECORD
  {}                                          -- no parameters
```

8.4.4.5 Write_Unforce_Variables

8.4.4.5.1 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 varialbles of this station have been unforced.

8.4.4.5.2 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
                                          -- one entry for each unforced
                                             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 number of elements
                                             (structured types) according
                                             to 6.2 (RTP)
    var_type          UNSIGNED8,          -- type of the variable
                                             according to 6.2 (RTP)
    var_offset        UNSIGNED16,         -- offset of the variable
    chk_offset        UNSIGNED16          -- offset of the check variable
                                             ('FFFF'H if unused).
}
  }
```

8.4.4.5.3 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
```

8.4.4.6 Write_Unforce_All

8.4.4.6.1 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.

8.4.4.6.2 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.
  }
```

8.4.4.6.3 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
```

8.4.4.7 Read_Variable_Bindings

8.4.4.7.1 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.

8.4.4.7.2 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
  {
  }
```

8.4.4.7.3 Reply_Read_Variable_Bindings

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="8">tnm_key</td><td colspan="8">sif_code = 38</td></tr>
<tr><td colspan="16">nr_vars</td></tr>
<tr><td colspan="16">variables_list: ARRAY[nr_vars] OF</td></tr>
<tr><td colspan="16">variable_name: STRING32</td></tr>
<tr><td colspan="8">(CHARACTER8)</td><td colspan="8">CHARACTER8 or ‘00’H</td></tr>
<tr><td colspan="8">var_properties</td><td colspan="8">individual_period</td></tr>
<tr><td colspan="16">standard_type</td></tr>
<tr><td colspan="4">bus_id</td><td colspan="12">port_address</td></tr>
<tr><td colspan="8">var_ size</td><td colspan="8">var_type</td></tr>
<tr><td colspan="16">var_offset</td></tr>
<tr><td colspan="16">chk_offset</td></tr>
</table>

```
Reply_Read_Variable_Bindings::= RECORD
  {
  nr_vars               UNSIGNED16,          -- number of variables in the
                                                list.
  variables_list        ARRAY [ nr_vars] OF
                                             -- one entry for each described
                                                variable
    {
    variable_name       STRING32,            -- local name of the variable
    var_properties      BITSET8
      {
      bnd     (0)                            -- 1 variable to bind
                                                0 no action
      phl     (1),                           -- 1 physical address (memory)
                                                0 logical address (port)
      reg     (6),                           -- 1 regular variable,
                                                0 maintenance variable
      imp     (7)                            -- 1 imported variable
                                                0 exported variable
      },
    individual_period   UNSIGNED8            -- individual period of the
                                                variable as power of 2 of 1
                                                ms
                                                (e.g. 4 = 16 ms).
    standard_type       ENUM16,              -- application-defined standard
                                                type.
    bus_id              UNSIGNED4,           -- traffic store to which
                                                variable is bound
    port_address        WORD12,              -- port address to which
                                                variable is bound
    var_size            UNSIGNED8,           -- size in bits (simple types)
                                                or number of elements
                                                (structured types) according
                                                to 6.2 (RTP)
    var_type            UNSIGNED8,           -- code of the type of the
                                                variable according to 6.2
                                                (RTP)
    var_offset          UNSIGNED16,          -- offset of the variable
                                                within the dataset
    chk_offset          UNSIGNED16           -- offset of the check variable
                                                (‘FFFF’H if unused).
    }
  }
```

8.4.4.8 Write_Variable_Bindings

8.4.4.8.1 Description

Binds or unbinds a number of variables to a specific traffic store and port address.

NOTE This service supports the ROSIN configuration.

8.4.4.8.2 Call_Write_Variable_Bindings

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 39							
nr_vars															
variables_list: ARRAY[nr_vars] OF															
variable_name: STRING32															
								CHARACTER8 or ‘00’H							
var_properties								standard_type							
individual_period															
bus_id				port_address											
var_size								var_type							
var_offset															
chk_offset															

```
Call_Write_Variable_Bindings::= RECORD
  {
  nr_vars               UNSIGNED16,         -- number of variables to be
                                               bound.
  variables_list        ARRAY [ nr_vars] OF
    {                                       -- array with one entry for
                                               each variable to be bound
    variable_name       STRING32            -- local name of the variable
                                               to be bound or unbound
    var_properties      BITSET8
      {
      bnd    (0)                            -- 1 variable to bind
                                               0 no action
      phl    (1),                           -- 1 physical address (memory)
                                               0 logical address (port)
      reg    (6),                           -- 1 regular variable,
                                               0 maintenance variable
      imp    (7)                            -- 1 imported variable
                                               0 exported variable
      },
    standard_type       ENUM8,              -- application-defined standard
                                               type
    individual_period   UNSIGNED16          -- individual period of the
                                               variable in milliseconds.
    standard_type       ENUM8,              -- application code of type
    bus_id              UNSIGNED4,          -- traffic store to which
                                               variable is bound, or
                                               unbound.
    port_address        WORD12,             -- port to which variable is
                                               bound (unbound if port is 0)
    var_size            UNSIGNED8,          -- size in bits (simple types)
                                               or number of elements
                                               (structured types) according
                                               to 6.2.
    var_type            UNSIGNED8,          -- type according to 6.2 (RTP)
    var_offset          UNSIGNED16,         -- offset of the variable
    chk_offset          UNSIGNED16,         -- offset of the check variable
                                               ('FFFF'H if unused).
    }
```

```
  }
```

8.4.4.8.3 Reply_Write_Variable_Bindings

0–7	8–15
tnm_key	sif_code = 39

```
Reply_Write_Variable_Bindings::= RECORD
  {
  }
```

8.4.4.9 Write_Attach_Port

8.4.4.9.1 Description

Attach ports of the Traffic_Store to specific inputs or outputs (Class 2 stations).

8.4.4.9.2 Call_Write_Attach_Port

0–3	4–7	8–15
tnm_key		sif_code = 29
nr_ports		
port_point_list: ARRAY[nr_vars] OF		
bus_id	port_address	
point		
filter		
gain		
offset		

```
Call_Write_Attach_Port::= RECORD
  {
  nr_ports             UNSIGNED16,         -- number of considered ports
  port_point_list      ARRAY [nr_ports] OF
    {                                      -- list of ports and points
                                              containing in each element
    ds_name            RECORD
      {
      bus_id           UNSIGNED4 (0..15),  -- bus_id (Traffic_Store) = (0
                                              by default)
      port_address     WORD12              -- 12-bit port address (shall
                                              be divisible by 2)
      }
    point_descriptor   RECORD              -- descriptor of the point,
                                              containing:
      {
      point            UNSIGNED16,         -- 16-bit identifier of the I/O
                                              point
      filter           UNSIGNED16,         -- filter time constant, in
                                              multiples of 10 ms (or 0 if
                                              unused)
      gain             UNSIGNED16,         -- value of the gain of an
                                              analog value
      offset           INTEGER16           -- value of the offset of an
                                              analog value
      }
    }
  }
```

8.4.4.9.3 Reply_Write_Attach_Port

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 39							

```
Reply_Write_Attach_Port::= RECORD
  {}                                         -- no parameters
```

8.4.5 Messages services

8.4.5.1 Read_Messenger_Status

8.4.5.1.1 Description

Retrieves the status of the messenger and its statistics counters.

8.4.5.1.2 Call_Read_Messenger_Status

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 40							

```
Call_Read_Messenger_Status::= RECORD
  {}                                         -- no parameters
```

8.4.5.1.3 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
  }
```

8.4.5.2 Write_Messenger_Control

8.4.5.2.1 Description

Sets parameters in the messenger.

8.4.5.2.2 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	pkr	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-Vz.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.
```

8.4.5.2.3 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
```

8.4.5.3 Read_Function_Directory

8.4.5.3.1 Description

Reads the Function_Directory at a Station.

8.4.5.3.2 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
```

8.4.5.3.3 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
    {                                     -- array of Function and
                                             Station pairs,  each
                                             consisting of:
    function_id       UNSIGNED8,          -- Function_Identifier
    station_id        UNSIGNED8           -- corresponding
                                             Station_Identifier
    }
  }
```

8.4.5.4 Write_Function_Directory

8.4.5.4.1 Description

Writes the Function_Directory at a Station.

8.4.5.4.2 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
    {                                      -- array of Function and
                                              Station pairs, each
                                              consisting of:
    function_id        UNSIGNED8,          -- Function_Identifier
    station_id         UNSIGNED8           -- corresponding
                                              Station_Identifier
    }
  }
```

8.4.5.4.3 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
```

8.4.5.5 Read_Station_Directory

8.4.5.5.1 Description

Reads the Station_Directory at a Station (if it exists).

8.4.5.5.2 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
```

8.4.5.5.3 Reply_Read_Station_Directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 44							
reserved1								nr_stations							
station_list: ARRAY [nr_stations] OF															
station_id								next_station_id							
bus_id								reserved2							
device_address															

```
Reply_Read_Station_Directory::= RECORD
  {
  reserved1             WORD8 (=0),          -- reserved
  nr_stations           UNSIGNED8,           -- number of stations in the
                                                list
  station_list          ARRAY [nr_stations] OF
    {                                        -- list of stations with the
                                                corresponding  Next_Station,
                                                consisting 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
    reserved2           WORD8 (=0),          -- reserved
    device_address      UNSIGNED16           -- address of the device which
                                                carries Next_Station
    }
  }
```

8.4.5.6 Write_Station_Directory

8.4.5.6.1 Description

Writes the Station_Directory at a Station (if it exists).

8.4.5.6.2 Call_Write_Station_Directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 45							
clear_directory								nr_stations							
station_list: ARRAY[nr_stations] OF															
station_id								next_station_id							
bus_id								reserved1							
device_address															

```
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
    {                                       -- list of stations with the
                                               corresponding Next_Station,
                                               consisting 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
    }
  }
```

8.4.5.6.3 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
```

8.4.5.7 Read_Group_Directory

8.4.5.7.1 Description

Reads the Group_Directory at a Station (if it exists).

8.4.5.7.2 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
```

8.4.5.7.3 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							
g7	g6	g5	g4	g3	g2	g1	g0	g15	g14	g13	g12	g11	g10	g9	g8
g23															g24
g39															g40
g55								g63							g56

```
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.
  }
```

8.4.5.8 Write_Group_Directory

8.4.5.8.1 Description

Writes the Group_Directory at a Station (if it exists).

8.4.5.8.2 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							
g7	g6	g5	g4	g3	g2	g1	g0	g15	g14	g13	g12	g11	g10	g9	g8
g23															g24
g39															g40
g55								g63							g56

```
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
}
```

8.4.5.8.3 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
```

8.4.5.9 Read_Node_Directory

8.4.5.9.1 Description

Reads the Node_Directory at a Node (if it exists).

8.4.5.9.2 Call_Read_Node_Directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 48							

```
Call_Read_Node_Directory::= RECORD
  {}                                       -- no parameters
```

8.4.5.9.3 Reply_Read_Node_Directory

0–7	8–15
tnm_key	sif_code = 48
reserved1	nr_nodes
nodes_list: ARRAY [nr_nodes] OF	
node_address	reserved2
device_address	

```
Reply_Read_Node_Directory::= RECORD
  {
  reserved1             WORD8 (=0),           -- reserved
  nr_nodes              UNSIGNED8,            -- number of Nodes in the list
  nodes_list            ARRAY [nr_nodes] OF
    {                                         -- list of nodes with the
                                                 corresponding
                                                 Device_Address, consisting
                                                 of:
    node_address        UNSIGNED8,            -- 8-bit Node_Address
    reserved2           WORD8 (=0),           -- reserved
    device_address      UNSIGNED16            -- Device_Address of the nodes
    }
  }
```

8.4.5.10 Write_Node_Directory

8.4.5.10.1 Description

Writes the Node_Directory at a Node (if it exists).

8.4.5.10.2 Call_Write_Node_Directory

0–7	8–15
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
    {                                         -- list of nodes with the
                                                 corresponding
                                                 Device_Address, consisting
                                                 of:
    node_address        UNSIGNED8,            -- 8-bit Node_Address
    reserved1           WORD8 (=0),           -- reserved
    device_address      UNSIGNED16            -- Device_Address of the nodes
    }
  }
```

8.4.5.10.3 Reply_Write_Node_Directory

0–7	8–15
tnm_key	sif_code = 49

```
Reply_Write_Node_Directory::= RECORD
  {}                                        -- no parameters
```

8.4.6 Domain services

8.4.6.1 Read_Memory

8.4.6.1.1 Description

This service reads in one operation several memory regions, each consisting of a number of identical, consecutive items, whose 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.

8.4.6.1.2 Call_Read_Memory

0–7	8–15
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
    {                                       -- list of regions, each region
                                               comprising
    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.
    }
  }
```

8.4.6.1.3 Reply_Read_Memory

0 … 7	8 … 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_octets           UNSIGNED16,          -- number of octets in the
                                                transported field, including
                                                padding octets.
    item_value_list     ARRAY ALIGN16 [nr_octets] OF
      {                                      -- array of [nr_octets] values,
                                                each comprising:
      item_value        WORD8                -- item values, transmitted in
                                                continuous order.
      }                                      -- the first octet shall be a
                                                padding octet if the
    }                                        -- region begins at an odd
                                                address.
  }
```

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.

8.4.6.2 Write_Memory

8.4.6.2.1 Description

This service writes in one operation several memory regions, each consisting of a number of identical items of a given size.

8.4.6.2.2 Call_Write_Memory

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
tnm_key	sif_code = 51
reserved1	nr_regions
region_list: ARRAY [nr_regions] OF	
base address	
nr_items	
reserved2	item_size
region_value_list: ARRAY [nr_regions] OF	
item_value_list: ARRAY [nr_items] OF	
first octet if even nr_octets else meaningless	first octet if odd nr_octets
even octets	odd octets
last octet (if nr_octets is odd)	last octet (or '00'H if nr_octets is odd)

```
Call_Write_Memory::= RECORD
  {
  reserved1             WORD8 (=0),          -- reserved
  nr_regions            UNSIGNED8,           -- number of regions to be
                                                written
  region_list           ARRAY[nr_regions) OF
    {                                        -- list of regions, for each
                                                region:
    base_address        UNSIGNED32,          -- base address of the region
                                                (may be odd).
    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
    {                                        -- array of region_values, each
                                                consisting of:
    item_value_list     ARRAY ALIGN16 [nr_items] OF
      {                                      -- array of nr_items values,
                                                each comprising:
      ONE_OF [item_size]                     -- transmitted in continuous
                                                order.
        {
        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.

8.4.6.2.3 Reply_Write_Memory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 51							

```
Reply_Write_Memory::= RECORD
  {}                                          -- no parameters
```

8.4.6.3 Download_Setup

8.4.6.3.1 Description

Download setup 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.

8.4.6.3.2 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.
    }
  }
```

8.4.6.3.3 Reply_Write_Download_Setup

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
    }
  }
```

8.4.6.4 Download_Segment

8.4.6.4.1 Description

This service transmits a segment of a defined size into the domain opened by Write_Download_Setup.

8.4.6.4.2 Call_Write_Download_Segment

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
    }
  }
```

8.4.6.4.3 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
```

8.4.7 Task services

8.4.7.1 Read_Tasks_Status

8.4.7.1.1 Description

This service retrieves the name and status of the tasks installed in a Station.

8.4.7.1.2 Call_Read_Tasks_Status

0–7	8–15
tnm_key	sif_code = 60

```
Call_Read_Tasks_Status::= RECORD
  {}                                        -- no parameters
```

8.4.7.1.3 Reply_Read_Tasks_Status

0–7	8–15
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
    }
  }
```

8.4.7.2 Write_Tasks_Control

8.4.7.2.1 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.

8.4.7.2.2 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
  }
```

8.4.7.2.3 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
```

8.4.8 Clock services

8.4.8.1 Read_Clock

8.4.8.1.1 Description

Reads the clock value at the selected Station.

8.4.8.1.2 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
```

8.4.8.1.3 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
  }
```

8.4.8.2 Write_Clock

8.4.8.2.1 Description

Sets the clock at the selected Station.

8.4.8.2.2 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
  }
```

8.4.8.2.3 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
```

8.4.9 Journal Service

8.4.9.1 Read_Journal

8.4.9.1.1 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.

8.4.9.1.2 Call_Read_Journal

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 80							
reserved1								number_entries							

```
Call_Read_Journal::= RECORD
  {
  reserved1              WORD8 (=0),          -- reserved
  number_entries         UNSIGNED8            -- up to 255 entries.
  }
```

8.4.9.1.3 Reply_Read_Journal

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 80							
reserved1								number_entries							
event_list ARRAY [number_entries] OF															
time_stamp: TIMEDATE48															
file_name: STRING16															
(CHARACTER8)								‘00’H							
line_number															
reserved2								event_type							
event_description: STRING78															
(CHARACTER8)								‘00’H							

```
Reply_Read_Journal::= RECORD
  {
  reserved1              WORD8 (=0),
  number_entries         UNSIGNED8,           -- number of returned entries
  event_list             ARRAY [number_entries] OF
    {
    time_stamp           TIMEDATE48,          -- time-stamp when the event
                                                 occurred
    file_name            STRING16,            -- as supplied by __FILE__ in
                                                 ANSI ’C‘ (null terminated
                                                 string)
    line_number          UNSIGNED16,          -- as supplied by __LINE__ in
                                                 ANSI ’C’
    reserved2            WORD8 (=0),
    event_type           ENUM8                -- event type
      {
      INFO             (0),
      WARNING          (1),
      ERROR            (2)
      },
    event_description  STRING78               -- event description (null
                                                 terminated string)
    }
  }
```

8.4.10 Equipment Service

8.4.10.1 Read_Equipment

8.4.10.1.1 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.

8.4.10.1.2 Call_Read_Equipment

0–7	8–15
tnm_key	sif_code = 82
reserved1	reserved2

```
Call_Read_Equipment::= RECORD
  {
  reserved1             WORD8 (=0),          -- reserved
  reserved2             WORD8 (=0),          -- reserved
  }
```

8.4.10.1.3 Reply_Read_Equipment

0–7	8–15
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
    }
  }
```

8.5 Interface Procedures

The interface procedures are split between a Manager Interface and an Agent Interface.

8.5.1 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_xxx.

Description	Calls a remote service.	

Syntax	`MM_RESULT`
	`UNSIGNED8`
	`const AM_ADDRESS*`
	`struct MM_CALL *`
		`mm_service_req`
		`(`
		`station_id;`
		`agent_adr;`
		`mm_call`
		`);`

Input	station_id	Station_Identifier of this Station
	agent_adr	Network_Address of the Agent
	mm_call	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 MM_RESULT, defined in 8.4.1.5.

Description	The service confirm procedure mm_service_conf returns the result of the service call to the Manager. This procedure may be a polled procedure or an indication procedure.	

Syntax	`MM_RESULT`
	`UNSIGNED8`
	`AM_ADDRESS *`
	`struct MM_REPLY *`
		`mm_service_conf`
		`(`
		`station_id;`
		`agent_adr;`
		`mm_reply`
		`);`

Output	station_id	Station_Identifier of this Station
	agent_adr	Network_Address of the Agent
	mm_reply	in case MM_RESULT 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 MM_RESULT, defined in 8.4.1.5.

8.5.2 Agent interface

8.5.2.1 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.

8.5.2.2 Agent control procedures

8.5.2.2.1 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	`MA_PERMISSION`
	
	`UNSIGNED8`
		`ma_ask_permission`
		`(`
		`task_id`
		`);`	

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

8.5.2.2.2 Procedure ma_give_permission

Description	The application responds to the change request with this procedure, indicating whether change is allowed or not.		

Syntax	`void`
	
	`ENUM8`
		`ma_give_permission`
		`(`
		`decision`
		`);`	

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	-		

8.5.2.3 User service subscription

8.5.2.3.1 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	`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 );`	
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.

Syntax:

```
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
                      );
```

8.5.2.3.2 Type MA_SERVICE_CLOSE

Description	Type declaration of a procedure to be called for closing a service.	
Syntax	`typedef void ( * MA_SERVICE_CLOSE ) (void);`	
Input	undefined	user-defined

Syntax:

```
typedef void          ( * MA_SERVICE_CLOSE )
                      (void);
```

8.5.2.3.3 Procedure ma_subscribe

Description	Indicates which user procedure to call when a user-defined service call is received. A previously assigned SIF_code is overwritten without a warning.	

Syntax	`MM_RESULT`
	`ENUM16`
	`ENUM8`
	`MA_SERVICE_CALL`
	`MA_SERVICE_CLOSE`
	`void *`
		`ma_subscribe`
		`(`
		`command;`
		`sif_code;`
		`service_call;`
		`service_close`
		`service_desc`
		`);`

Input	command	0: subscribe 1: desubscribe
	sif_code	user SIF_code (≥ 128)
	service_call	procedure variable of type MA_SERVICE_CALL which will execute the service when called.
	service_close	procedure which the Agent will call when the Reply_Message has been completely sent (e.g. to free the buffer).
	service_desc	descriptor of the service, as a visible string terminated by a ‘00’H character.
Return	MM_RESULT	

8.5.2.4 Restart procedure subscription

8.5.2.4.1 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	`typedef void`	`( * MA_STATION_RESTART )` `( );`

8.5.2.4.2 Procedure ma_subscribe_restart

Description	Indicates which user procedure to call when the Station is reset or a reservation time-out takes place.	

Syntax	`MM_RESULT`
	`MA_STATION_RESTART`
		`ma_subscribe_restart`
		`(`
		`station_restart`
		`);`

Input	station_restart	procedure which the Agent will call when the reservation time-out elapses or when a reset command has been received.
Return	MM_RESULT	

Bibliography

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

ISO/IEC 8482, *Information technology – Telecommunications and information exchange between systems – Twisted pair multipoint interconnections*

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

MATÉRIEL ÉLECTRONIQUE FERROVIAIRE – RÉSEAU EMBARQUÉ DE TRAIN (TCN) –

Partie 2-1: Bus de Train Filaire (WTB)

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) La CEI elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de la CEI. La CEI n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de la CEI peuvent faire l'objet de droits de brevet. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale CEI 61375-2-1 a été établie par le comité d'études 9 de la CEI: Matériels et systèmes électriques ferroviaires.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
9/1642/FDIS	9/1666/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Une liste de toutes les parties de la série CEI 61375, présentées sous le titre général *Matériel électronique ferroviaire – Réseau embarqué de train (TCN)*, peut être consultée sur le site web de la CEI.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de la CEI sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

Cette première édition annule et remplace les articles de la deuxième édition de la CEI 61375-1 publiée en 2007 applicables à la spécification du MTB dont elle constitue une révision technique.

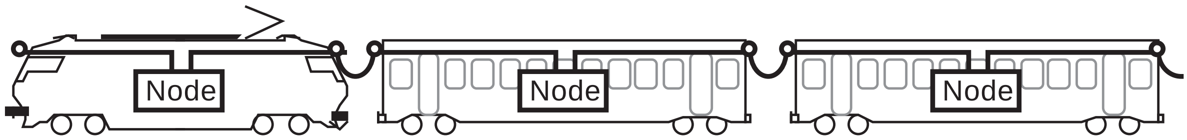
Elle a été établie en tenant compte de la troisième édition de la CEI 61375-1.

IMPORTANT – Le logo "colour inside" qui se trouve sur la page de couverture de cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.

INTRODUCTION

La présente partie de la CEI 61375 spécifie un composant du Réseau Embarqué de Train, le Bus de Train Filaire (WTB pour Wire Train Bus), un bus série de transmission de données destiné principalement, mais pas exclusivement, à l'interconnexion des rames qui font l'objet d'accouplement et de désaccouplement fréquents, comme pour les trains internationaux UIC.

La Figure 1 illustre l'application WTB.



Légende

Anglais	Français
Node	Nœud

Figure 1 – Bus de Train Filaire

La présente norme définit ces interfaces en tant que raccordements à un réseau de communication de données, appelé Réseau Embarqué de Train (TCN, pour Train Communication Network).

Le Réseau Embarqué de Train a une structure hiérarchisée avec deux niveaux de réseaux, un Réseau Central de Train et un Réseau de Rame:

a) pour relier les rames de trains à composition variable (voir définition) tels que les trains internationaux UIC, la présente norme spécifie un Bus de Train appelé Bus de Train Filaire (WTB, pour Wire Train Bus);
b) pour relier des équipements standards embarqués (un Réseau de Rame, par exemple) le Bus de Véhicule Multifonctions (MVB, pour Multifunction Vehicle Bus) peut être utilisé.

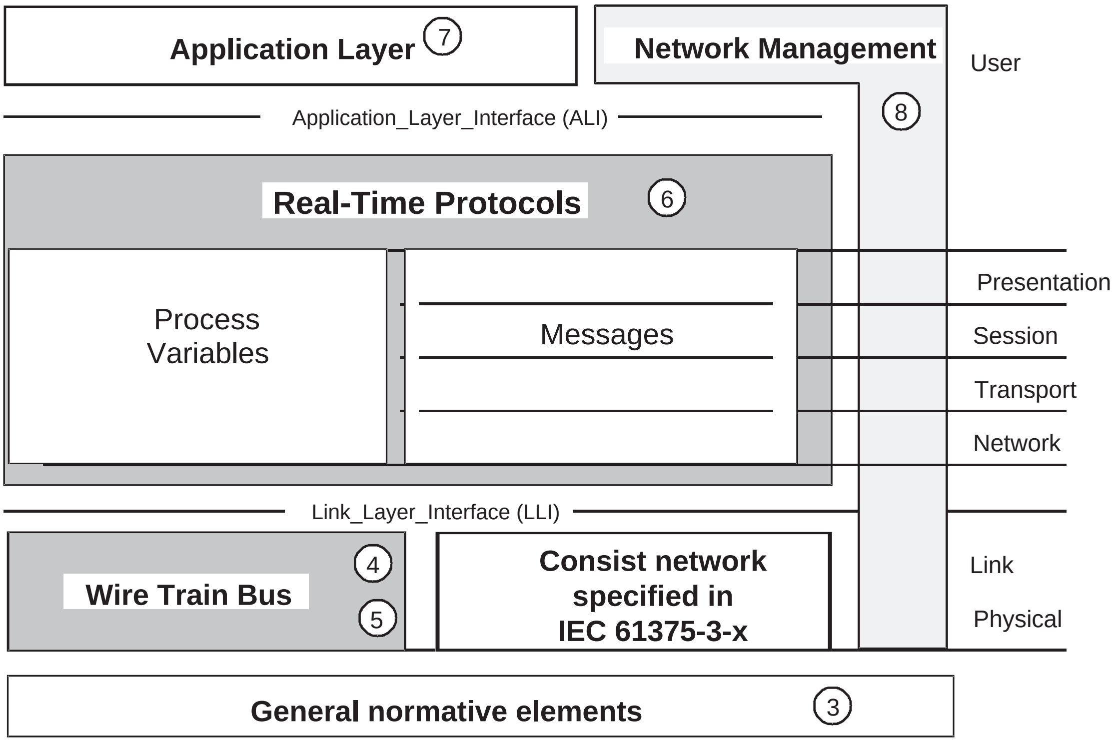
Dans l'architecture TCN, le WTB utilise des Protocoles en Temps Réel, qui offrent deux services de communication:

c) le service de Variables de Processus, offert par une base de données distribuée, mise à jour en temps réel et périodiquement par diffusion;
d) le service de Messagerie, offrant le transfert de messages à la demande, sous forme:
   - de messages point à point et/ou
   - de messages publipostés.

Dans le TCN, le WTB offre une gestion commune de réseau qui permet le déverminage, la mise en service et la maintenance sur tout le réseau.

Le MVB du Réseau de Rame partage des Protocoles en Temps Réel et une Gestion de Réseau avec le WTB. Une autre mise en œuvre des Réseaux de Rame doit être adaptée aux Protocoles en Temps Réel et à la Gestion de Réseau du WTB.

Le TCN présente une structure similaire à celle du modèle de Système Ouvert d'Interconnexion défini dans l'ISO/CEI 7498-1 (voir la Figure 2).



NOTE Les chiffres cerclés se réfèrent aux articles de la présente norme.

Légende

Anglais	Français
Application layer	Couche Application
Application Layer Interface	Interface de Couche d'Application
Link Layer Interface	Interface de Couche de Liaison
Network management	Gestion de Réseau
Real-time protocols	Protocoles en Temps Réel
User	Utilisateur
Process variables	Variables de processus
Messages	Messages
Presentation	Présentation
Session	Session
Transport	Transport
Network	Réseau
Wire train bus	Bus de Train Filaire
Consist network specified in IEC 61375-3-x	Réseau de Rame spécifié dans la CEI 61375-3-x
General normative elements	Eléments normatifs généraux

Figure 2 – Stratification du TCN

Pour des raisons rédactionnelles, la présente norme a été divisée en six articles:

Article 1 – Domaine d'application

Article 2 – Références normatives

Article 3 – Termes et définitions, abréviations, conventions

Articles 4 et 5 – Bus de train filaire (WTB)

– Contrôle de la couche physique et de la couche de liaison

Article 6 – Protocoles en Temps Réel

- Variables: Interface de couche de liaison et interface de couche d’application
- Messages: Interface de couche de liaison, Protocoles, interface de couche d’application
- Représentation des données

Article 7 – Couche d’application

- Triage des Données de Processus
- Détection de l’emplacement de défaut en ligne du WTB

Article 8 – Gestion de Réseau de Train

- Configuration, supervision et commande du réseau

MATÉRIEL ÉLECTRONIQUE FERROVIAIRE – RÉSEAU EMBARQUÉ DE TRAIN (TCN) –

Partie 2-1: Bus de Train Filaire (WTB)

1 Domaine d'application

La présente partie de la CEI 61375 s'applique à la communication de données dans les Trains à Composition Variable, c'est-à-dire qu'elle couvre la communication de données aussi bien entre les rames que dans les rames desdits trains à composition variable.

L'application de la présente norme au bus de communication de données (WTB) permet l'interopérabilité des différentes rames d'un train à composition variable dans le trafic international. Le bus de communication de données dans les rames (le MVB, par exemple) est donné comme solution recommandée pour fonctionner avec ledit TCN. Dans tous les cas, le fournisseur aura à faire la preuve de la compatibilité entre le WTB et le Réseau de Rame proposé.

Après accord entre acheteur et fournisseur, la présente norme peut s'appliquer en outre aux trains indéformables et aux automotrices.

NOTE 1 Pour la définition des Trains à Composition Variable, Automotrices et Trains Indéformables, voir Article 3.

NOTE 2 Les véhicules routiers comme les bus et les trolleybus ne sont pas traités dans la présente norme.

2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

CEI 60571, *Equipements électroniques utilisés sur les véhicules ferroviaires*

CEI 60807 (toutes les parties), *Connecteurs rectangulaires utilisés aux fréquences inférieures à 3 MHz*

CEI 61375-1, *Matériel électronique ferroviaire – Réseau embarqué de train (TCN) – Partie 1: TNC – Réseau embarqué de train – Architecture générale*

CEI 61375-2-2:2012, *Matériel électronique ferroviaire – Réseau embarqué de train (TCN) – Partie 2-2: Bus de train filaire – Essais de conformité*

CEI 61375-3-1, *Matériel électronique ferroviaire – Réseau embarqué de train (TCN) – Partie 3-1: Bus de Véhicule Multifonctions (MVB)*

ISO/CEI 8802-2, *Technologies de l'information – Télécommunications et échange d'informations entre systèmes – Réseaux locaux et métropolitains – Exigences spécifiques – Partie 2: Contrôle de liaison logique*

ISO/CEI 8824 (toutes les parties), *Technologie de l'information – Notation de syntaxe abstraite numéro un (ASN.1)*

ISO/CEI 8825 (toutes les parties), *Technologie de l'information – Règles de codage ASN.1*

ISO/CEI 8859-1, *Technologies de l'information – Jeux de caractères graphiques codés sur un seul octet – Partie 1: Alphabet latin n° 1(disponible en anglais seulement)*

ISO/CEI 9646 (toutes les parties), *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Cadre général et méthodologie des tests de conformité*

ISO/CEI 10646, *Technologie de l'information – Jeu universel de caractères codés sur plusieurs octets (JUC)*

ISO/CEI 13239, *Technologies de l'information – Télécommunications et échange d'information entre systèmes – Procédures de commande de liaison de données à haut niveau (HDLC)*

UIT-T Recommandation V.24, *Liste des définitions des circuits de jonction entre l'équipement terminal de traitement de données (DTE) et l'équipement de terminaison du circuit de données (DCE)*

UIT-T Recommandation Z.100, *Langage de description et de spécification (SDL)*

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

CODE UIC 556, *Transmission d'information dans le train (bus de train)*

CODE UIC 557, *Technique de diagnostique dans les voitures*

3 Termes et définitions, abréviations, conventions

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions suivants s'appliquent.

NOTE La première lettre de chaque mot-clé de la présente norme est écrite en majuscule et, lorsque celui-ci se compose de deux ou plusieurs mots, ces derniers sont reliés par un trait bas. Cette convention permet de retrouver les mots-clés dans les documents.

3.1.1
adresse
identifie une entité de communication. Il en existe plusieurs sortes selon la couche considérée

3.1.2
agent
processus d'application localisé dans une Station. L'agent accède aux objets gérés localement sous le contrôle d'un gérant

3.1.3
Données Apériodiques
transmission de Données de Processus suite à une requête. Ce service n'est pas utilisé

3.1.4
Couche Application
couche supérieure du modèle OSI en interface directe avec l'Application

3.1.5
Interface de Couche Application
définition des services offerts par la Couche Application

3.1.6
Adaptateur de Messagerie d'Application
code directement appelé par l'application réalisant les services de Messagerie

3.1.7
Interface de Messagerie d'Application
définition des services de Messagerie

3.1.8
Processus d'Application
entité de communication, réalisée par exemple par une tâche

3.1.9
Processeur d'Application
processeur faisant tourner un Processus d'Application communiquant

3.1.10
Interface de Supervision d'Application
définition des services de Supervision disponibles en particulier pour l'Agent

3.1.11
Adaptateur de Variables d'Application
code appelé directement par l'Application réalisant les services de Variables

3.1.12
Interface de Variables d'Application
définition des services de Variables

3.1.13
arbitre
dispositif, ou protocole commun suivi par plusieurs dispositifs, qui sélectionne l'un des dispositifs en conflit pour la position de maître

3.1.14
Canal Auxiliaire
canal utilisé pour détecter des Nœuds supplémentaires

3.1.15
Période de Base
l'activité du bus est divisée en périodes. La plus courte est la Période de Base, qui se compose d'une Phase Périodique (pour les Données Périodiques) et d'une Phase Apériodique (pour les Données de Messagerie et les Données de Supervision)

3.1.16
big-endian (gros-boutiste)
ordonnancement des données pour leur stockage et leur transmission dans lequel la part de poids fort d'une donnée représentée sur plusieurs octets est stockée à l'adresse la plus faible et transmise en premier

3.1.17
bit-stuffing (bourrage de bits)
méthode préconisée par l'ISO/CEI 13239 pour éviter le risque de confusion entre les Données de Trame et un Délimiteur. Cette méthode consiste à insérer un "0" supplémentaire après chaque séquence de cinq "1" consécutifs, et à supprimer ce "0" lors de la réception

3.1.18
pont
dispositif qui stocke et retransmet les trames d'un bus à l'autre en fonction de leurs adresses de Couche de Liaison

3.1.19
diffusion
transmission presque simultanée des mêmes informations vers plusieurs destinataires. La diffusion sur TCN n'est pas considérée comme fiable (certains destinataires pourraient recevoir l'information et d'autres pas)

3.1.20
bus
support de communication qui diffuse la même information quasiment au même instant à tous les équipements connectés, ce qui permet à tous les dispositifs d'avoir une vue identique de son état, au moins dans le cas d'un arbitrage

3.1.21
Contrôleur de Bus
processeur ou circuit intégré en charge de la Couche de Liaison

3.1.22
Commutateur (Contacteur) de Bus
commutateur ou relais d'un Nœud de WTB qui connecte électriquement les sections de câble en provenance des deux directions

3.1.23
Appelant (Caller)
Processus d'Application qui initie un échange de message

3.1.24
code Détecteur d'Erreur
méthode de détection d'erreurs (de transmission) reposant sur l'ajout aux données utiles d'une somme de contrôle ou d'un contrôle de redondance cyclique (CRC) calculé sur les données utiles

3.1.25
Validant/Variable de Contrôle
Variable de Processus de type booléen antivalent protégeant une autre Variable de Processus

3.1.26
Check Offset
Offset (décalage) en bit d'une Variable de Contrôle dans un Dataset

3.1.27
Train Indéformable
train composé d'un ensemble de rames, dont la composition n'est pas modifiée en exploitation normale, par exemple, le métro, le train de banlieue ou les rames à grande vitesse

3.1.28
composition
numéro et caractéristiques des rames composant un train

3.1.29
configuration
définition de la topologie d'un bus, des équipements qui lui sont connectés, de leurs capacités et du trafic qu'ils génèrent. Par extension, l'opération consistant à charger dans les équipements les informations de configuration avant leur passage en mode opérationnel

3.1.30
Confirmation de Connexion
réponse du Consommateur à la Demande de Connexion émise par le Producteur

3.1.31
Demande de Connexion
premier paquet d'un message envoyé d'un Producteur vers un Consommateur

3.1.32
Rame
véhicule unique ou groupe de véhicules qui ne sont pas séparés en exploitation normale. Une Rame peut contenir aucun, un seul ou plusieurs Réseaux de Rame

3.1.33
réseau de Rame
bus connectant des équipements dans une rame (le MVB, par exemple) et qui se conforme ou s'adapte aux Protocoles en Temps Réel TCN comme décrit dans le présent document

3.1.34
cohérence
Dataset composé de plusieurs éléments qui est considéré comme cohérent si tous les éléments sont lus ou écrits au cours d'une opération indivisible

3.1.35
Consommateur
destinataire d'un message au niveau de la Couche Transport (voir: Producteur)

3.1.36
rame de continuité
rame dépourvue de Nœud de Bus de Train en fonctionnement, mais assurant la continuité du Bus de Train entre les deux rames adjacentes

3.1.37
conversation
échange de données au niveau de la Couche Application, consistant en un Message d'Appel et un Message de Réponse (ce dernier est inexistant dans le cas du protocole de distribution). Une conversation commence par la première trame Connect Request (Demande de Connexion) et cesse quand le dernier acquittement pour le Message de Réponse a été reçu, ou quand il n'est plus attendu

3.1.38
datagramme
trame contenant toutes les informations nécessaires à son acheminement vers son destinataire final, sans connaissance du contenu de la trame précédente. Les Datagrammes n'utilisent aucun mécanisme préliminaire d'établissement de connexion et ne sont pas acquittés au niveau de la Couche de Liaison

3.1.39
Dataset
ensemble des Variables de Processus transmises dans une trame de Données de Processus

3.1.40
délimiteur
séquence incluant des symboles de violation de code (ni "1", ni "0") qui est utilisée pour délimiter le début (Délimiteur de Début) et la fin (Délimiteur de Fin) de chaque trame, comme défini par exemple dans la CEI 61158-2

3.1.41
Dispositif Destinataire
destinataire d'une trame au niveau de la Couche de Liaison (voir: Dispositif Emetteur)

3.1.42
dispositif
module connecté à un (ou plusieurs) bus

3.1.43
Adresse de Dispositif
identifie un dispositif sur un bus. Une Adresse de Dispositif est codée sur 8 bits sur WTB, les 6 bits de poids faible correspondant à l'Adresse de Nœud;

Un dispositif connecté à plusieurs bus peut avoir une Adresse de Dispositif différente pour chaque bus. Des dispositifs particuliers comme les répéteurs participent uniquement à la Couche Physique et n'ont pas d'Adresse de Dispositif

3.1.44
Direction 1
une direction d'un Nœud WTB

3.1.45
Direction 2
autre direction d'un Nœud WTB

3.1.46
Délimiteur de Fin de Trame
séquence qui termine la trame avant que le support ne retourne au repos

3.1.47
Nœud d'Extrémité
nœud qui réalise la terminaison des deux segments de bus auquel il est connecté mais qui n'établit pas la continuité entre eux

3.1.48
Tour d'Evénements
séquence d'interrogation au cours de laquelle tous les événements en attente au démarrage du tour sont traités

3.1.49
boîte d'extension
élément de connectique dans lequel le câble principal est interrompu et prolongé de manière passive par un câble d'extension vers un dispositif

3.1.50
câble d'extension
câble permettant l'insertion d'un Nœud dans un câble principal. Il est composé de deux paires torsadées par ligne, dont la surface de section est plus petite que le câble principal lui-même

3.1.51
dispositif de terrain
équipement connectant des capteurs et actionneurs simples à un bus à l'extérieur d'un panier

3.1.52
final
destinataire d'un paquet (données ou acquittement) au niveau de la Couche Réseau. Quand deux équipements communiquent sur le même bus, le final se trouve dans le Dispositif Destinataire (voir: origine)

3.1.53
Drapeau
séquence de symboles "0" ou de "1" qui sert à délimiter le début ou la fin d'une trame. Un drapeau qui apparaîtrait dans les données à transmettre peut être modifié par le bourrage de bits, comme défini par exemple dans l'ISO/CEI 13239

3.1.54
trame
séquence de symboles consécutifs envoyée en une fois par un émetteur, entre deux périodes d'inactivité de la ligne

3.1.55
Code de Detection d'Erreurs
somme de Contrôle (FCS pour Frame Check Sequence) de 16 bits telle que définie dans l'ISO/CEI 13239

3.1.56
Données de Trame
données transmises entre le Préambule et le Délimiteur de Fin (sur le WTB)

3.1.57
fritting (nettoyage des contacts)

application d'une tension aux bornes d'un contact destinée à nettoyer son éventuelle oxydation

3.1.58
Fonction
Processus d'Application qui échange des messages avec un autre Processus d'Application

3.1.59
Répertoire (Table) de Fonctions
répertoire qui associe un Indicatif de Fonction à un Indicatif de Station, et inversement

3.1.60
Indicatif de Fonction
mot de 8 bits identifiant une Fonction

3.1.61
F code
dans une Trame Maître, indique le type de demande et la taille de la Trame Esclave attendue en réponse

3.1.62
passerelle
connexion de différents bus au niveau de la Couche Application requérant une analyse des données dépendant de l'application ainsi qu'une conversion de protocole

3.1.63
Adresse de Groupe
adresse d'un groupe de distribution auquel appartient un Nœud

3.1.64
Répertoire de Groupe
répertoire indiquant à un Nœud le groupe de distribution auquel il appartient

3.1.65
distance de Hamming
nombre minimal de bits d’une séquence de bits correcte donnée qui, si inversée, crée une séquence de bits erronée ne pouvant être distinguée d’une séquence correcte

3.1.66
HDLC
High-level Data Link Control (Commande de Liaison de Données à Haut Niveau), un ensemble de protocoles normalisés incluant l'ISO/CEI 13239 relative à la transmission de données

3.1.67
Données HDLC
données transmises dans une trame HDLC

3.1.68
Inauguration
opération entreprise en cas de modification de la composition, qui donne à tous les Nœuds du WTB leur position relative par rapport au Maître, leur orientation et le descripteur de tous les Nœuds nommés présents sur le même bus

3.1.69
Période Individuelle
période séparant deux transmissions consécutives de la même Donnée de Processus par la même source. La Période Individuelle est un multiple d’une puissance de deux de la Période de Base

3.1.70
instance
a) l’un parmi plusieurs objets qui partagent la même définition (instance d'objet)

b) l’une parmi plusieurs exécutions (simultanées ou non) d'un même programme (instance de processus)

3.1.71
intégrité
capacité d'un système à reconnaître et à éliminer des données erronées en cas de dysfonctionnement de ses sous-ensembles

3.1.72
Nœud Intermédiaire
nœud qui établit la continuité entre les deux sections de bus qui lui sont connectées, mais qui n'assure pas leur terminaison

3.1.73
câble de jonction
câble connectant les câbles principaux de deux rames consécutives, éventuellement de section supérieure à celle du câble principal, et qui est connecté à la main dans le cas d'un câble UIC. Il y a en général deux câbles de jonction entre les rames

3.1.74
Ligne
bus non redondant. Un bus redondant (dual-thread) met en œuvre deux lignes

3.1.75
Unité de Ligne
ensemble des composants assurant la connexion électrique à la ligne

3.1.76
Adresse de Liaison
adresse utilisée par la Couche de Liaison pour identifier le Bus et l'Adresse de Dispositif émetteur ou destinataire d'un paquet

3.1.77
Contrôle de Liaison
champ de la trame HDLC qui identifie le type de la trame

3.1.78
Données de Liaison
données véhiculées par la Couche de Liaison mais non interprétées par elle

3.1.79
En-tête de liaison
champ d'une trame de Données de Messagerie interprété par la Couche de Liaison

3.1.80
Couche de Liaison
couche du modèle OSI permettant d'assurer des transferts de données en mode point à point ou en mode diffusion, entre différents équipements situés sur le même bus

3.1.81
Interface de Couche de Liaison
interface entre la Couche de Liaison et les couches de communication supérieures

3.1.82
Gestion de Couche de Liaison
interface contrôlant la Couche de Liaison pour sa gestion

3.1.83
little-endian (petit-boutiste)
ordonnancement des données pour leur stockage et leur transmission dans lequel la part de poids faible d'une donnée représentée sur plusieurs octets est stockée à l'adresse la plus faible et transmise en premier

3.1.84
réseau local
portion de réseau caractérisé par un mode d'accès au support et par un espace d'adressage uniques

3.1.85
contrôle de liaison logique
protocole et formats de trame associés qui servent à contrôler la Couche de Liaison

3.1.86
Adresse Logique
adresse qui n'est pas liée à un équipement particulier (adresse de Données de Processus, par exemple)

3.1.87
Port Logique
port d'un équipement utilisé pour le trafic de Données de Processus et adressé par une Adresse Logique

3.1.88
Macro Cycle
nombre de Périodes de Base correspondant à une Macro Période

3.1.89
Macro Période
période Individuelle la plus longue mesurée en millisecondes. A l'issue de la Macro Période, le trafic périodique reprend selon le même enchaînement

3.1.90
Canal Principal
canal utilisé pour la transmission sur le bus principal

3.1.91
Message de Gestion
message échangé entre un Gestionnaire et un Agent dans le cadre de la Gestion de Réseau

3.1.92
Gestionnaire
fonction d'une Station dédiée à la Gestion de Réseau et qui envoie des Messages d'Appel de gestion en utilisant les Adresses Système

3.1.93
triage
attribution de noms ou d'adresses aux Variables de Processus d'un dataset qui, sur le WTB, dépend du Type de Nœud et de sa Version

3.1.94
Maître
équipement qui envoie de manière spontanée des informations sur un bus à destination de plusieurs dispositifs esclaves. Il peut autoriser un Esclave à transmettre une Trame Esclave uniquement dans un certain laps de temps

3.1.95
Trame Maître
trame transmise par un Maître

3.1.96
Délimiteur de Début de Trame Maître
Délimiteur de Début d'une Trame Maître

3.1.97
contrôle d'accès au support
sous-couche de la Couche de Liaison qui contrôle l'accès au support (arbitrage, transfert de maîtrise, interrogation)

3.1.98
interface dépendante du support
interface mécanique et électrique entre le support de transmission et le Dispositif de Connexion au Support

3.1.99
support
milieu de propagation du signal: câbles électriques, fibres optiques, etc.

3.1.100
Dispositif de Connexion au Support
dispositif dont la fonction est d'assurer la connexion d'un équipement au support

3.1.101
message
données transmises en un ou plusieurs paquets

3.1.102
Messages (Messagerie)
service de transmission du TCN

3.1.103
Données de Messagerie
données transmises de manière apériodique au niveau de la Couche de Liaison pour assurer le service de transmission de messages; le service Couche de Liaison correspondant

3.1.104
messager
pile de communication assurant la transmission de messages de bout en bout, ainsi que l'interface avec l'application

3.1.105
distribution (publipostage)
transmission du même message à un groupe de Destinataires identifiés par leur Adresse de Groupe. Le terme « distribution » est utilisé même si le groupe comprend tous les Destinataires

3.1.106
Bus de Véhicule Multifonctions
MVB (Multifunction Vehicle Bus)
Bus de Rame à utiliser pour relier des stations programmables et de simples capteurs/actionneurs

3.1.107
Train à Unités Multiples
train composé d'un ensemble de rames indéformables, la composition de cet ensemble pouvant évoluer en exploitation normale

3.1.108
réseau
ensemble de différents systèmes possibles de communication qui échangent de l'information selon un moyen communément accepté

3.1.109
Adresse Réseau
adresse qui identifie une Fonction ou une Station sur le réseau. Il peut s'agir d'une Adresse Utilisateur ou d'une Adresse Système

3.1.110
En-tête Réseau
partie d'une trame de Données de Messagerie relevant de la Couche Réseau

3.1.111
Couche Réseau
couche du modèle OSI assurant le routage entre les différents bus

3.1.112
Gestion de Réseau
opérations nécessaires à la configuration, la supervision, le diagnostic et la maintenance à distance du réseau

3.1.113
Nœud
équipement sur le Bus de Train Filaire, qui sert de passerelle entre le Bus de Train et le réseau de Rame

3.1.114
Adresse de Nœud
adresse d'un Nœud sur le Bus de Train (6 bits). Il s'agit des 6 bits de poids faible de l'Adresse de Dispositif codée sur 8 bits sur le WTB

3.1.115
Descripteur de Nœud
structure de données de 24 bits permettant de coder la Période de Nœud et la Clé de Nœud d'un Nœud

3.1.116
Répertoire (Table) de Nœud
répertoire de correspondance entre l'Adresse de Nœud et l'Adresse de Dispositif (correspondance biunivoque sur le WTB)

3.1.117
Clé de Nœud
indicatif de 16 bits fourni par l'application permettant d'identifier le type et la version d'un Nœud. Le Maître le distribue à tous les autres Nœuds après chaque changement de la composition et avant d'échanger des données

3.1.118
Période de Noeud
sur le WTB, Période Individuelle requise pour un Nœud (identique à la Période Individuelle sauf en cas de surcharge du réseau)

3.1.119
octet
mot de 8 bits stocké dans la mémoire ou transmis sous forme d'unité [*]

3.1.120
Train à Composition Variable
train composé d'un ensemble de rames, dont la configuration peut changer en exploitation normale, par exemple les trains internationaux UIC

3.1.121
origine
expéditeur d'un paquet (donnée ou acquittement) au niveau de la Couche Réseau. Lorsque deux équipements communiquent sur le même bus, l'Origine est située sur dispositif émetteur (voir: final)

3.1.122
paquet
unité de message (information, acquittement ou commande) transmise dans exactement une trame de Données de Messagerie

---

[*] La CEI préconise "octet" au lieu de "byte".

3.1.123
période
laps de temps après lequel le trafic sur le bus se répète selon le même motif

3.1.124
Données Périodiques
Données de Processus transmises de manière périodique, à une fréquence correspondant à leur Période Individuelle

3.1.125
Liste Périodique
liste des Nœuds, adresses ou équipements à interroger à chaque période de Macro Cycle

3.1.126
Phase Périodique
phase durant laquelle le Maître gère le transfert des Données Périodiques selon le contenu de sa Liste Périodique

3.1.127
Adresse Physique
adresse de Nœud sur le WTB qui identifie des équipements communiquant sur le même bus

3.1.128
Port Physique
port utilisé pour le transfert des Données de Messagerie ou des Données de Supervision, adressé par l'Adresse de Dispositif

3.1.129
écartement
distance entre deux équipements adjacents sur le même bus électrique pour éviter les accumulations de charge sur ledit bus

3.1.130
interrogation
envoi d'une Trame Maître pour recevoir une Trame Esclave

3.1.131
Port
structure mémoire qui contient des données en émission ou en réception et dans laquelle une nouvelle valeur écrase la valeur précédente (registre par opposition à queue). Le bus et la ou les applications peuvent accéder simultanément à un Port

3.1.132
Table d'Index de Port
table de conversion établissant la correspondance entre l'adresse mémoire d'un port et l'Adresse Logique des Données de Processus

3.1.133
Préambule
séquence de signaux débutant une trame dont la fonction est de permettre la synchronisation du récepteur utilisé sur le WTB

3.1.134
Couche Présentation
couche du modèle OSI en charge de la représentation et de la conversion des données

3.1.135
Données de Processus
données identifiées par leur source diffusées de manière périodique par la couche de liaison en relation avec la transmission de Variables de Processus; le service Couche de Liaison correspondant

3.1.136
Variable de Processus
variable exprimant l'état d'un processus (vitesse, commande de frein, par exemple)

3.1.137
Producteur
émetteur d'un message au niveau de la Couche Transport (voir: Consommateur)

3.1.138
Editeur
source d'un Dataset en diffusion (voir: Souscripteur)

3.1.139
PV Name
indicatif d'une Variable de Processus

3.1.140
PV Set
ensemble de Variables de Processus appartenant au même Dataset

3.1.141
queue
mémoire stockant un ensemble ordonné de trames selon une stratégie premier arrivé, premier servi

3.1.142
panier
équipement contenant un ou plusieurs dispositifs, associé au même segment

3.1.143
réassemblage
action consistant à reconstituer un long message à partir de plusieurs paquets générés par une segmentation

3.1.144
récepteur
dispositif électronique qui peut recevoir des signaux en provenance du support physique

3.1.145
Queue de Réception
queue de réception de Données de Messagerie sur un dispositif

3.1.146
fonctionnement normal
activité normale du bus par opposition au mode Inauguration (WTB)

3.1.147
répéteur
interconnexion entre segments de bus au niveau de la Couche Physique permettant une extension du bus au-delà des limites permises par des composants passifs. Les segments

interconnectés fonctionnent à la même vitesse et selon le même protocole. Le retard introduit par un répéteur est de l'ordre d'un temps bit

3.1.148
Répondeur
Processus d'Application à qui l'Appelant transmet un Message d'Appel et qui répond par un Message de Réponse

3.1.149
taux d'erreur résiduel
probabilité d'atteinte à l'intégrité (bit erroné non reconnu) par bit transmis

3.1.150
routeur
interconnexion entre deux bus au niveau de la Couche Réseau, le transfert de datagrammes d'un bus sur l'autre s'effectuant sur la base de leur Adresse Réseau

3.1.151
balayage
interrogation des dispositifs selon une séquence donnée à des fins de supervision

3.1.152
section
portion d'un segment connectée passivement à une autre section sans terminaison intermédiaire

3.1.153
segment
portion de câble à laquelle les dispositifs sont connectés, terminée à ses deux extrémités par son impédance caractéristique. Les segments peuvent être composés de plusieurs sections (non terminées) connectées par des connecteurs

3.1.154
segmentation
division d'un long message en plusieurs trames de longueur plus courte lors de sa transmission

3.1.155
Queue de Transmission
queue d'émission de Données de Messagerie au sein d'un dispositif

3.1.156
service
capacités et moyens d'un sous-système (une couche de communication, par exemple) offerts à un utilisateur

3.1.157
En-tête de Session
partie d'une trame de Données de Messagerie relevant de la Couche Session

3.1.158
Couche Session
couche du modèle OSI en charge de l'établissement et de la terminaison des communications

3.1.159
Flanc A
un flanc d'une rame par rapport à un Nœud du WTB

3.1.160
Flanc B
autre flanc d'une rame par rapport à un Nœud du WTB

3.1.161
Esclave
dispositif qui reçoit ou envoie des informations en provenance ou sur le bus en réponse à une requête (aussi appelée interrogation) émise par un Maître

3.1.162
Trame-Esclave
trame envoyée par un Esclave

3.1.163
Dispositif Emetteur (source)
émetteur d'une trame au niveau de la Couche de Liaison (voir: dispositif destinataire)

3.1.164
transmission spontanée
transmission réalisée sur demande, lorsqu'un événement extérieur au réseau le requiert (aussi appelée transmission apériodique, sur événement, sur demande)

3.1.165
Données Apériodiques
trames de données transmises sur demande pour transmettre des Données de Messagerie ou des Données de Supervision

3.1.166
Phase Apériodique
deuxième partie de la Période de Base, dédiée à la transmission de messages sur demande et de données de gestion de bus

3.1.167
coupleur en étoile
dispositif recevant la lumière en provenance d'une fibre optique et la retransmettant sur plusieurs autres fibres

3.1.168
Station
dispositif capable de communiquer des messages, par opposition aux dispositifs simples, et prenant en charge une Fonction Agent

3.1.169
Répertoire (Table) de Station
répertoire qui établit la correspondance entre Indicatif de Station et Adresse de Liaison et inversement

3.1.170
Indicatif de Station
mot de 8 bits identifiant une Station

3.1.171
Mot d'Etat de la Station
descripteur de 16 bits définissant l'état et les capacités d'une Station

3.1.172
Maître Fort
Nœud Fort qui est Maître actif et n'y renonce pas tant qu'il n'a pas été rétrogradé en Nœud Faible

3.1.173
Nœud Fort
nœud sélectionné par l'application pour devenir Maître Fort. Il ne peut exister qu'un Maître Fort par segment de bus

3.1.174
branche
connexion en T à partir d'une ligne de bus électrique (au niveau de l'embranchement) permettant de connecter un dispositif à la ligne.

3.1.175
Souscripteur
un des destinataires d'un Dataset en diffusion (voir: Éditeur)

3.1.176
Données de Supervision
données transmises sur un bus uniquement dans le cadre de la supervision de la Couche de Liaison (inauguration sur le WTB, par exemple)

3.1.177
Adresse Système
Adresse Réseau d'un Message de Gestion échangée entre un Gestionnaire et un Agent, consistant en une Adresse de Nœud et un Indicatif de Station

3.1.178
embranchement
endroit où un segment est connecté. Un embranchement est une fourche électrique à trois voies

3.1.179
télégramme
Trame Maître et Trame Esclave correspondante considérées comme un tout

3.1.180
terminaison
circuit qui termine une ligne de transmission électrique, de manière idéale sur son impédance caractéristique

3.1.181
Interrupteur (Commutateur) de Terminaison
interrupteur qui insère la Terminaison en bout de segment sur le WTB

3.1.182
Topographie
structure de données décrivant les Nœuds connectés au Bus de Train, incluant leur adresse, orientation, position et Descripteur de Nœud

3.1.183
topologie
type d'interconnexion de câble et nombre de dispositifs qu'un réseau donné prend en charge

3.1.184
Compteur de Topographie
compteur dans un Nœud incrémenté lors de chaque nouvelle Inauguration

3.1.185
Traffic Store (Mémoire Partagée)
mémoire partagée accessible tant par le réseau que par l'utilisateur, et qui contient le Port de Données de Processus

3.1.186
Réseau Embarqué de Train (TCN – Train Communication Network)
réseau de communication de données permettant de connecter des équipements électroniques programmables embarqués sur des véhicules ferroviaires

3.1.187
Bus de Train, Réseau Central de Train
bus connectant les rames d'un train, en particulier le WTB, et qui est conforme aux protocoles TCN

3.1.188
Gestion de Réseau de Train (TNM – Train Network Management)
services de Gestion de Réseau du TCN

3.1.189
émetteur-récepteur
association d'un émetteur et d'un récepteur

3.1.190
émetteur
dispositif électronique permettant de transmettre un signal sur le support physique

3.1.191
Données de Transport
données transportées par la Couche Transport, mais non interprétées par celle-ci

3.1.192
En-tête de Transport
partie d'une trame de Données de Messagerie relevant de la Couche Transport

3.1.193
Couche Transport
couche du modèle OSI responsable du contrôle de flux de bout en bout et de la reprise sur erreur

3.1.194
câble principal
câble circulant le long des rames, par opposition au câble d’extension ou au câble de jonction

3.1.195
Adresse Utilisateur
Adresse Réseau d'un Message Utilisateur échangé entre Fonctions, composée d’une Adresse de Nœud (ou Adresse de Groupe) et un Indicatif de Fonction

3.1.196
Message Utilisateur
message échangé entre des Fonctions utilisateur

3.1.197
Variables
service de transmission du TCN

3.1.198
Var_Offset
offset (décalage) en bit d'une Variable de Processus dans un Dataset

3.1.199
Descripteur de Véhicule
informations descriptives d'un véhicule particulier dépendant de l'application (sa longueur et son poids, par exemple)

3.1.200
Maître Faible
Nœud Faible qui est Maître actif et prêt à abandonner son rôle à un Maître Fort s'il s'en présente un

3.1.201
Nœud Faible
Nœud qui peut prendre la maîtrise du bus spontanément, mais qui l'abandonne si un Noeud Fort se manifeste

3.1.202
Bus de Train Filaire (WTB – Wire Train Bus)
Bus de Train conçu pour des rames fréquemment accouplées et désaccouplées, comme par exemple les trains internationaux UIC

3.2 Abréviations

ALI	Application Layer Interface (Interface de Couche Application), la définition de la sémantique de tous les services réseau utilisés par l'application (un ensemble de primitives, exprimées en tant que procédures, constantes et types de données)
AMA	Application Messages Adapter (Adaptateur de Messagerie d'Application), code directement appelé par l'application qui met en œuvre le service de Messagerie
AMI	Application Messages Interface (Interface de Messagerie d'Application), définition des services de message
ANSI	American National Standard Institute, organisme de normalisation des États-Unis
ASI	Application Supervision Interface (Interface de Supervision d'Application), définition des services de Gestion
ASN.1	Abstract Syntax Notation Number 1, concernant la présentation des données (ISO/CEI 8824)
AVA	Application Variables Adapter (Adaptateur de Variables d'Application), code directement appelé par l'application mettant en œuvre les services de Variables de Processus
AVI	Application Variables Interface (Interface de Variables d'Application), définition des services de Variables de Processus
BER	Basic Encoding Rules (Règles de Codage de Base), syntaxe de transfert pour les types de données ASN.1 (ISO/CEI 8825)
BR	Bit Rate (Débit Binaire), débit de passage des données sur le support, exprimé en bits par seconde (bit/s) ou en hertz (Hz), selon ce qui est approprié

BT	Bit Time (Temps bit), durée de la transmission d'un bit, exprimée en µs
UIT	Union Internationale des Télécommunications, organisme de normalisation international pour les télécommunications dont le siège est à Genève
CRC	Cyclic Redundancy Check (Contrôle par Redondance Cyclique), vérification de l'intégrité des données par division en polynômes
DIN	Deutsches Institut für Normung, organisme de normalisation allemand
EIA	Electronics Industries Association, organisme de normalisation des Etats-Unis
EP	Câble de frein électro-pneumatique tel qu'il est décrit dans la fiche 648 de l'UIC
ERRI	European Railway Research Institute, laboratoire dont le siège est à Utrecht, Pays-Bas
FCS	Frame Check Sequence (Séquence de Contrôle de Trame), code de détection d'erreur ajouté aux données transmises, tel que spécifié dans l'ISO/CEI 13239
HDLC	High-level Data Link Control (Commande de Liaison de Données à Haut Niveau), protocole Couche de Liaison dont le format de trame est défini dans l'ISO/CEI 13239
CEI	Commission Electrotechnique Internationale, Genève
IEEE	Institute of Electrical and Electronics Engineers, New York
ISO	International Standard Organisation, Organisation Internationale de Normalisation, Genève
LFLD	Line Fault Location Detection, détection de l’emplacement de défaut en ligne
LLC	Logical Link Control (Contrôle de Liaison Logique), sous-couche de la Couche de Liaison gérant l'échange de données
LME	Layer Management Entity (Entité de Gestion de Couche), entité en charge de la supervision d'une couche pour la Gestion de Réseau
LMI	Layer Management Interface (Interface de Gestion de Couche), services fournis par la LME
MAC	Medium Access Control (Contrôle d’Accès au Support), sous-couche au sein de la Couche de Liaison décidant quel dispositif est autorisé à émettre vers le bus
MAU	Medium Attachment Unit (Dispositif de Connexion au Support), partie d'un Nœud qui assure l'interface électrique vers le bus et qui fournit/accepte des signaux logiques binaires
MIB	Management Information Base (Base d’Informations de Gestion), ensemble de tous les objets auxquels la Gestion de Réseau accède
MVB	Multifunction Vehicle Bus (Bus de Véhicule Multifonctions), Réseau de Rame
NRZ	Non-Return to Zero, schéma d'encodage le plus simple dans lequel un bit est représenté par un niveau pour un "1" et un autre niveau pour un "0", ou inversement avec une synchronisation séparée
ORE	Office de Recherche et d'Essais, laboratoire UIC dont le siège est à Utrecht, Pays-Bas
OSI	Open System Interconnection (Interconnexion de Systèmes Ouverts), modèle de communication universel défini dans l’ISO/CEI 7498
PDM	Process Data Marshalling (Triage des Données de Processus)

PICS Protocol Implementation Conformance Statement (Déclaration de Conformité d'une Mise en œuvre de Protocole), définie dans l'ISO/CEI 9646

PTA Process Data to Traffic Store Adapter (Adaptateur des Données de Processus au Traffic Store), composant qui accède à un des Traffic Stores (Mémoires Partagées)

RIC Règlement pour l'emploi réciproque des voitures et des fourgons en trafic international, édité par l'UIC

RTP Real-Time Protocols (Protocoles en Temps Réel), protocoles de communication communs spécifiés à l'Article 6 de la présente norme

SDL Specification and Description Language (Langage de description et de spécification), langage de spécification défini par l'UIT-T Z.100 Annexe D pour les protocoles de communication

TCN Train Communication Network (Réseau Embarqué de Train), ensemble de réseaux de rame communicants et de Réseaux Centraux de Train

TNM Train Network Management (Gestion de Réseau de Train)

UIC Union Internationale des Chemins de Fer, association internationale des exploitants de chemins de fer

WTB Wire Train Bus (Bus de Train Filaire)

3.3 Conventions

3.3.1 Base des valeurs numériques

La présente norme utilise une représentation décimale pour toutes les valeurs numériques, sauf mention contraire.

Les valeurs analogiques et fractionnaires comportent une virgule.

EXEMPLE 1: La tension est de 20,0 V.

Les valeurs binaires et hexadécimales sont représentées en convention ASN.1 (ISO/CEI 8824).

EXEMPLE 2: Le nombre décimal 20 est codé sur 8 bits comme: '0001 0100'B = '14'H

3.3.2 Conventions d'appellation

La première lettre de chaque mot-clé utilisé dans les spécifications du TCN est une majuscule.

Lorsque le mot est composé, les différentes parties de l'appellation sont séparées par un espace.

Lorsqu'une structure de données est associée à un mot-clé, le type de ladite structure se compose des mêmes mots de base séparés par un trait bas.

Lorsque la valeur correspondant au mot-clé est transmise dans un message, le champ correspondant porte le même nom que le type de la structure, mais en minuscule.

Lorsque la valeur est transmise comme paramètre, le paramètre porte le même nom que le champ dans un message.

Dans les diagrammes SDL, la variable correspondante porte le même nom que le type de la structure, mais sans trait bas.

EXEMPLES

Topo Counter est un compteur de la couche de liaison.

Il est du type Topo_Counter, qui est un UNSIGNED6.

Lorsque sa valeur est transmise dans un message, le champ correspondant s'appelle 'topo_counter'.

Lorsque sa valeur est transmise par une interface de procédure, le paramètre s'appelle 'topo_counter', son type en 'C' est Type_Topo_Counter.

Dans les diagrammes SDL, la variable représentant le compteur s'appelle TopoCounter.

3.3.3 Conventions pour les valeurs de temps

Les valeurs de temps qui commencent par une minuscule (t mm, par exemple) sont des intervalles de temps mesurables.

Les valeurs de temps qui commencent par une majuscule (T reply, par exemple) sont des paramètres ou des valeurs de temporisation.

3.3.4 Conventions pour les interfaces de procédure

Une interface de procédure est définie par un ensemble de primitives de service qui représentent une interaction abstraite, indépendante de la mise en œuvre, entre l'utilisateur du service et le fournisseur de service.

Dans la présente norme, ces primitives sont exprimées en tant que procédures dans la syntaxe ANSI C avec des types de paramètres.

Il convient de considérer qu'il s'agit uniquement d'une description sémantique qui n'implique pas une mise en œuvre ou un langage particuliers. Toute interface qui fournit la même sémantique est autorisée.

La conformité à la syntaxe de cette interface ne peut pas être revendiquée. Les mises en œuvre peuvent librement modifier les noms de procédure ou de paramètres pour ajouter des paramètres ou scinder des procédures, dans la mesure où le service spécifié est fourni.

Les procédures d'interface sont définies dans la syntaxe ANSI C en utilisant la police Courrier.

Les noms de procédures, les variables et les paramètres apparaissent tous en minuscule.

EXEMPLE 1: lm_send_request

Les constantes et les définitions de type apparaissent toutes en majuscule.

EXEMPLE 2: UNSIGNED32

Le nom d'une procédure ou d'un type se voit ajouter un préfixe:

pour le service de Variables:

- lp_ ou LP_ Couche de Liaison
  ap_ ou AP_ Couche Application

pour le service de Messagerie:

- MD_ messages en général
  lm_ ou LM_ Couche de Liaison
  nm_ ou NM_ Couche de Réseau
  tm_ ou TM_ Couche de Transport
  sm_ ou SM_ Couche Session
  am_ ou AM_ Couche Application

Le Tableau 1 montre un modèle utilisé pour les procédures et les types.

Tableau 1 – Modèle pour la spécification d'une procédure d'interface

Définition	*Le type de service ou de données est exprimé ici.* *Dans le cas d'une procédure d'indication, l'événement qui déclenche l'appel est indiqué ici, en commençant par "When" ("Quand")* *Le nom et les paramètres de la procédure de service sont définis ici.* *Dans le cas d'une procédure d'indication, le type de la procédure est spécifié.* *Les paramètres d'entrée, de sortie et de retour sont différenciés.*	

Syntaxe	`MD_RESULT lm_send_request`
	`( /* example */`
	`unsigned, destination,`
	`UNSIGNED8 link_control,`
	`MD_PACKET * p_packet`
	`ENUM8 * status`
	`)`	

Entrée	*Les paramètres d'entrée sont fournis à la procédure, qui n'est pas autorisée à les modifier.*	
	destination	Le type de donnée "unsigned" dépend du compilateur
	link_control	paramètre passé par référence, non modifié par la procédure. Le type de donnée est un mot en 8 bits.
	p_packet	"*" indique un pointeur vers une structure de données p_packet, du type MD_PACKET, défini ailleurs dans la présente norme
Sortie	*Il est prévu que l'appel modifie les paramètres.*	
	status	Le type ENUM8 est un type d'énumération 8-bit.
Résultat	*Le paramètre Result est un paramètre de sortie facultatif qui exprime le succès ou l'échec de l'appel, mais pas nécessairement du service.*	
	MD_RESULT	Le paramètre Result est un paramètre du type: AM_RESULT pour l'AMI, MD_RESULT pour le LMI, LP_RESULT pour le LPI, AP_RESULT pour l'AVI, Le modèle spécifie les codes d'erreur prévisibles pour chaque procédure individuellement. Le paramètre Result n'est pas décrit de manière explicite si les deux seules valeurs attendues sont: xx_OK = 0 achevé avec succès; xx_FAILURE <> 0 problèmes rencontrés. Le résultat peut également être retourné comme paramètre de sortie dans la liste des paramètres, selon la mise en œuvre.
Usage	Les règles répertoriées après le modèle de procédure indiquent comment il convient d'utiliser la procédure. Bien que les règles d'usage ne soient pas obligatoires, ne pas les suivre produit des résultats imprévisibles.	
NOTE Les structures de données représentées dans ce Tableau sont des spécifications d'interface qu'il convient de ne pas confondre avec le format des mêmes structures de données lorsque celles-ci sont transmises par l'intermédiaire d'un bus (voir 3.3.5).		

3.3.5 Spécification des données transmises

Le format des données transmises, que ce soit des trames isolées ou des messages complets, est spécifié de deux manières:

a) une forme graphique, qui n'est pas normative, mais qui montre intuitivement la structure du message;

b) une forme textuelle basée sur ASN.1, dont les règles de codage sont spécifiées en 6.3.

EXEMPLE 1: La forme graphique d'un message est donnée dans le Tableau 2, la forme textuelle correspondante est donnée dans le Tableau 3.

Tableau 2 – Exemple de structure de message

	premier octet transmis								**octet suivant transmis**							
bit->	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	snu	gni	node_id						station_id							
2	next_station_id								rsv1	tvd	topo_counter					
4	tnm_key								sif_code (= 3)							
6	parameter 1															
8	parameter 2								parameter 3						par4	
	parameter 5: ARRAY [n] OF (répéter le champ suivant n fois)															
	parameter 5.1															
	parameter 5.2															
	parameter5.3: STRING32															
	(CHARACTER8)								dernier caractère, sinon 0							

^octet

La numérotation de bit suit la représentation de valeur de puissance de deux dans un octet ou un mot. La numérotation de bit n'indique pas la séquence de transmission sur le bus, qui peut être différente: MSB d'abord ou LSB d'abord.

Dans la forme graphique, une ligne est employée pour chaque mot de 16 bits, sauf dans l'Article 5 (Contrôle de la Couche de Liaison WTB), où les lignes sont de huit bits.

Les tableaux de paramètres sont précédés d'un cadre de répétition par-dessus et sur le côté gauche.

Les répétitions peuvent être imbriquées (voir paramètre 5.3 dans le Tableau 2).

Si la taille d'un paramètre peut excéder trois mots, trois lignes sont accordées au paramètre, celle du milieu ayant une bordure grisée.

Tableau 3 – Exemple de forme de message textuel (correspondant au Tableau 2)

```
Call_Mgt_Message::=RECORD
 {
 snu                  BOOLEAN1 (=1)      -- '1' signifie que le message
                                            emploie une adresse système
 gni                  BOOLEAN1 (=0)      --  '0' signifie que l’adresse
                                            finale est un dispositif
                                            individuel
 node_id              UNSIGNED6          -- adresse du noeud
                                            d’extrémité à 6 bits
 station_id           UNSIGNED8          -- indicatif à 8 bits de la
                                            station
 next_station_id      UNSIGNED8          -- indicatif à 8 bits de la
                                            station suivante
 rsv1                 BOOLEAN1 (=0)      -- ce bit est toujours ‘0’
 tvd                  BOOLEAN1           -- ce bit indique que le
                                            compteur de topographie est
                                            valide
 topo_counter         UNSIGNED6          -- compteur de topographie de
                                            6 bits
 tnm_key              UNSIGNED8,         -- annonce un message d’appel
                                            de gestion de réseau
 sif_code             UNSIGNED8,         -- il existe un code SIF
                                            différent pour chaque
                                            Message de Gestion
 parameter1           INTEGER16,         -- valeur de 16 bits. Si le
                                            paramètre comporte moins de
                                            16 bits, la valeur est
                                            justifiée à droite et
                                            étendue par son signe (par
                                            exemple, si un seul octet
                                            est transmis, il l’est
                                            comme deuxième octet).
 parameter2           INTEGER8,          -- cette valeur est transmise
                                            dans la partie de poids le
                                            plus fort d'un mot.
 parameter3           UNSIGNED6          -- cette valeur est transmise
                                            dans l'octet de poids
                                            faible, mais les deux bits
                                            inférieurs de cet octet
                                            sont réservés pour le
                                            parameter4.
 par4                 ANTIVALENT2        -- parameter4 comporte deux
                                            bits
 parameter5           ARRAY [n] OF       -- parameter5 représente des
                                            données structurées à
  {                                      -- répéter n fois, contenant:
  parameter5.1        INTEGER16          -- premier paramètre du champ
                                            répété
  parameter5.2        UNIPOLAR4.16       -- deuxième paramètre du champ
                                            répété
  parameter5.3        STRING32           -- le troisième paramètre est
                                            une chaîne (jusqu’à
                                            32 caractères de 8 bits);
                                            -terminée par un ‘0’, ou
                                            par deux ‘0’ pour un
                                            alignement sur une limite
                                            d’un mot à 16 bits,
                                            – une chaîne vide est
                                            composée de
                                            32 caractères‘0’;
```

```
                                              - la taille réelle d’une
                                              chaîne est déduite du
                                              nombre de caractères
                                              significatifs qui précèdent
                                              le zéro.
  },
 }
```

Les noms de champs et leur type commencent respectivement par une minuscule et une majuscule. Parfois, le même type est utilisé comme format de transmission, auquel cas seule la première lettre est une majuscule, et de type C, auquel cas, le type complet est en majuscule.

EXEMPLE 2: Am_Result (format de transmission) et AM_RESULT (type C d’une procédure d’interface).

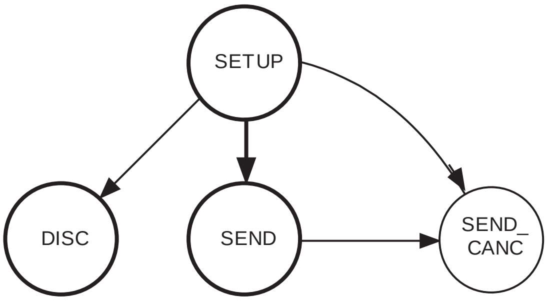
3.3.6 Conventions de diagrammes d'état

La machine d'état de protocole de transport est décrite dans l'ISO/CEI 8802-2 (Couche de Liaison Logique) sous la forme d'un tableau qui spécifie les transitions entre les états possibles dans lesquels une machine d'état peut se trouver.

Les transitions entre les états sont régies par les événements venant de la Couche Réseau (paquets d'entrée), de la Couche Session (commandes) ou des temporisations.

Une action dépendant de l'événement est exécutée avant de quitter l'état. Cette action définit l'état suivant.

La Figure 3 présente un exemple de diagramme de transition d'état.



Légende

Anglais	Français
SETUP	REGLAGE
DISC	DISQUE
SEND	ENVOI

Figure 3 – Exemple de transition d'état

A partir de ‘’SETUP ’’ (« RÉGLAGE »), la machine peut aller vers trois états différents, DISC (DISQUE), SEND (ENVOI) ou SEND_CANC (ANN_ENVOI).

La transition entre ces états est régie comme indiqué dans le Tableau 4.

Tableau 4 – Tableau de transitions d'état

État Actuel	Événement	Action(s)	État suivant (si > actuel)
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 ou
			SEND_CANC

	TMO AND (rep_cnt = MAX_REP_CNT)	close_send (AM_CONN_TMO_ERR);	DISC

Conformément au Tableau 4, trois événements génèrent une transition de l'état SETUP à l'état DISC:

a) rcv_DR (Disconnect_Request reçu, un événement réseau). L'action correspondante consiste à fermer la connexion (clode_send) avant de passer à l'état DISC;

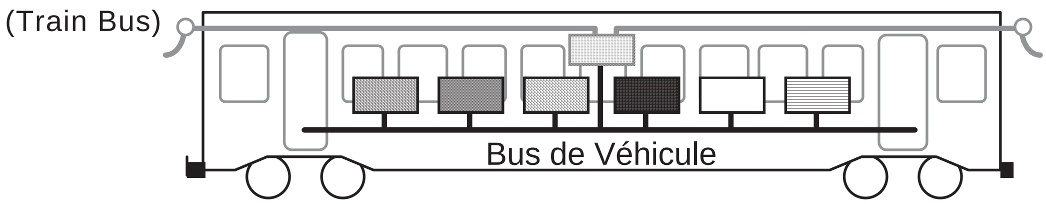
b) un autre événement réseau (Connect_Confirm reçu avec référence correcte), conduisant soit à l'état DISC, SEND ou SEND_CANC, selon le résultat de la procédure send_data ou d'annulation;

c) une valeur de temporisation conditionnée par le prédicat (rep_cnt = MAX_REP_CNT) qui provoque également la fermeture de la connexion.

3.4 Considérations générales

3.4.1 Interface entre équipements

La présente norme définit l'interface de communication de données des équipements d'une rame comme une connexion des dispositifs à un réseau de Rame (voir la Figure 4).



Légende

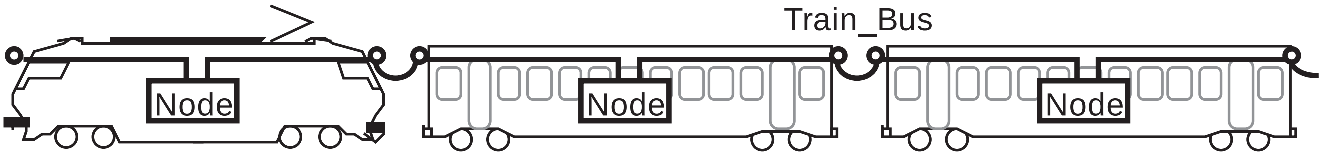
Anglais	Français
Train bus	Bus de train

Figure 4 – Interfaces entre équipements

Un Réseau de Rame est conçu principalement, mais pas exclusivement, pour l'interconnexion des équipements quand l'interopérabilité et l'interchangeabilité sont requises.

3.4.2 Interface entre rames

La présente norme définit l'interface de communication de données entre rames, comme la connexion de Nœuds situés dans les rames à un Bus de Train (voir la Figure 5).



Légende

Anglais	Français
Train bus	Bus de train
Node	Nœud

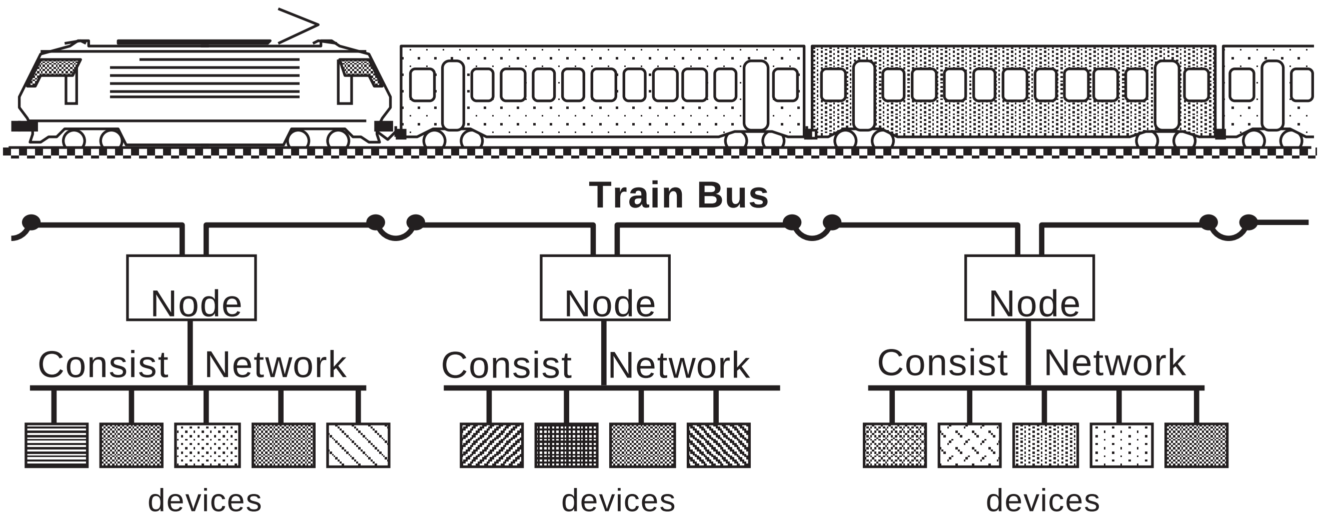
Figure 5 – Interfaces entre rames

En tant que Bus de Train, la présente norme spécifie le WTB, un bus en série de communication de données conçu principalement, mais pas exclusivement, pour l'interconnexion des rames de Train à Composition Variable.

NOTE Pour la définition des Trains à Composition Variable, voir Article 3.

3.4.3 Protocoles en Temps Réel

La présente norme définit l'architecture du TCN comme une hiérarchie à deux niveaux, un Bus de Train et un réseau de Rame (voir la Figure 6).



Légende

Anglais	Français
Train Bus	Bus de Train
Node	Nœud
Consist Network	Réseau de Rame
devices	dispositifs

Figure 6 – Bus de Train et réseau de Rame

En tant que protocoles de communication, l'Article 6 de la présente norme spécifie les Protocoles en Temps Réel (RTP) utilisés par tous les nœuds du WTB. Les dispositifs du réseau de Rame peuvent utiliser le même RTP (MVB, par exemple) ou adapter le protocole du réseau de Rame au RTP des nœuds WTB.

Le RTP spécifie l'Interface d'Application que le TCN fournit, composé de deux services fondamentaux: les Variables et les Messages.

Le RTP spécifie les protocoles de transmission qui interviennent dans des cheminements particuliers, le contrôle de débit et la reprise sur erreur.

Le RTP spécifie l'interface que les bus sont réputés fournir aux protocoles de transmission, et en particulier les deux services fondamentaux:

a) la diffusion périodique de Données de Processus adressées par la source; et

b) la transmission apériodique, sans connexion, de Données de Messagerie.

3.4.4 Gestion de Réseau

En tant que Gestion de Réseau, l'Article 8 de la présente norme spécifie la Gestion de Réseau TCN (TNM) comme un ensemble de messages échangés entre le Gestionnaire et l'Agent pour fournir les services fondamentaux.

3.4.5 Configurations

La présente norme peut être utilisée en partie ou comme un tout. Par exemple, il est possible d'utiliser:

a) le WTB sans réseau de rame ou avec un autre réseau de rame que le MVB;

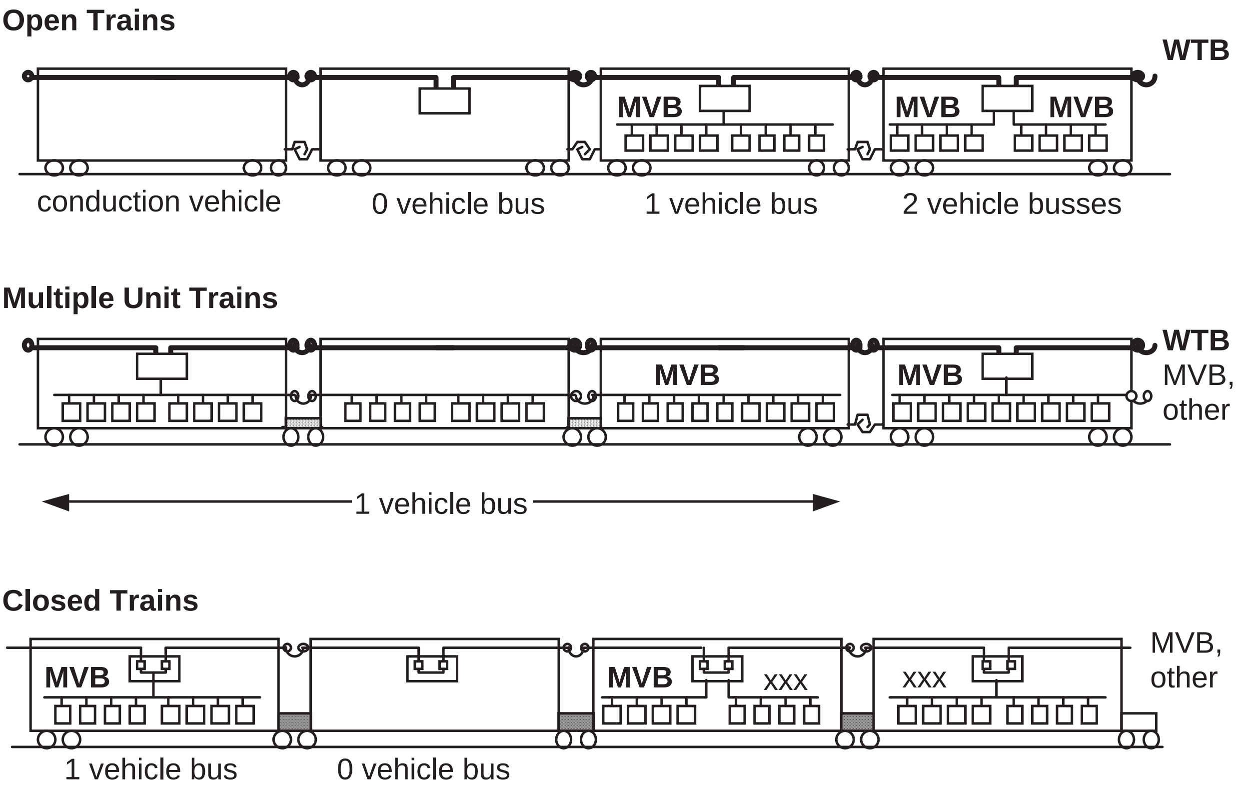
b) le RTP avec d'autres bus que le WTB ou le MVB.

La Figure 7 illustre trois configurations correspondant à trois domaines d'application:

c) la configuration des Trains à Composition Variable montre un Train à Composition Variable (un train UIC, par exemple) qui requiert une configuration automatique. Le WTB est utilisé comme Bus de Train standard, prenant en charge jusqu'à 32 Nœuds. Il peut y avoir 0, 1 ou davantage de Nœuds par rame. Il est possible d'associer jusqu'à 15 Réseaux de Rame (MVB ou autres) à chaque Nœud;

d) la configuration du Train à Unité Multiple représente deux trains indéformables connectés. Lorsque ces trains indéformables sont couplés et découplés fréquemment, le WTB peut être utilisé comme un Bus de Train standard. Mais lorsque d'autres moyens de configuration sont possibles, d'autres bus tels que le MVB peuvent être utilisés. Le réseau de Rame peut couvrir plusieurs véhicules;

e) la configuration des Trains Indéformables montre un train indéformable, dans lequel le réseau de Rame (MVB, par exemple) peut être utilisé à la fois comme Bus de Train et réseau de Rame.



Légende

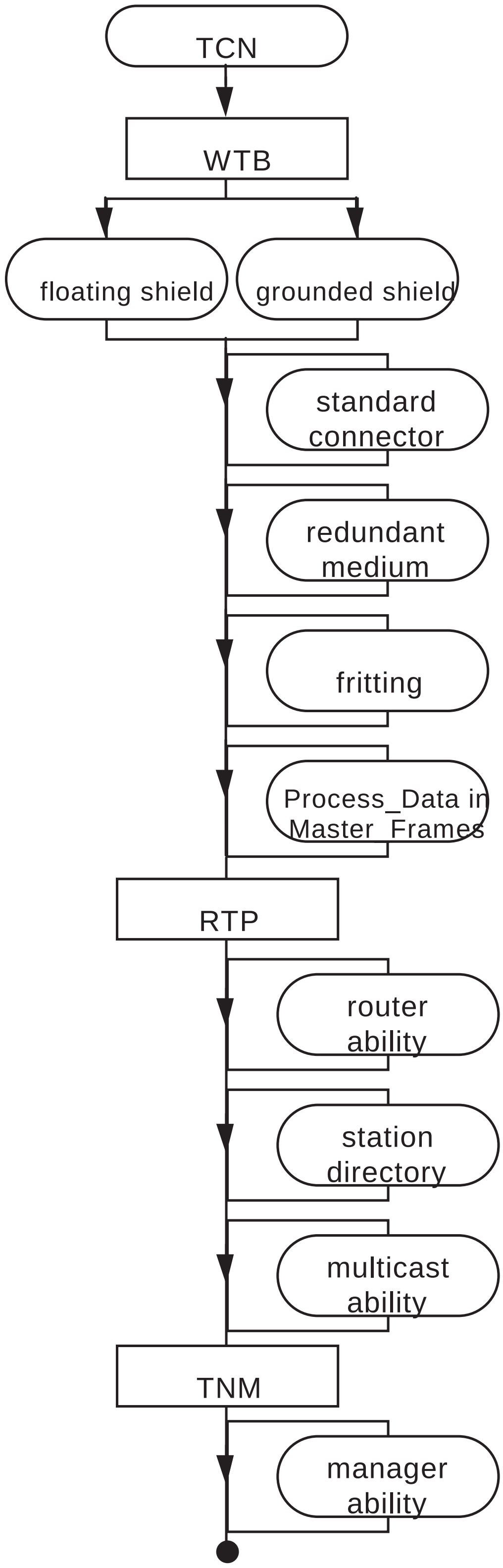
Anglais	Français
Open Trains	Trains à Composition Variable
conduction vehicle	véhicule pilote
vehicle bus	bus de véhicule
Multiple Unit Trains	Trains à Unité Multiple
other	autre
Closed Trains	Trains Indéformables

Figure 7 – Configurations du TCN

Dans les trois configurations de la Figure 7, le MVB du réseau de Rame est utilisé pour connecter les équipements embarqués, mais d'autres bus peuvent être utilisés comme réseau de Rame.

3.4.6 Structure d'un dispositif normalisé

Le présent paragraphe spécifie des options permises dans un dispositif TCN. Ces options sont décrites dans les articles correspondants et résumées dans la Figure 8.


TCN
WTB
floating shield
grounded shield
standard connector
redundant medium
fritting
Process_Data in Master_Frames
RTP
router ability
station directory
multicast ability
TNM
manager ability

Légende

Anglais	Français
floating shield	blindage flottant
grounded shield	blindage à la masse
standard connector	connecteur standard
redundant medium	support redondant
fritting	nettoyage des contacts
Process_Data in Master_Frames	Process_Data dans Master_Frames
RTP	RTP
router ability	capacité du routeur
station directory	répertoire de stations
multicast ability	capacité de distribution
manager ability	capacité de gestionnaire

Figure 8 – Options de configuration du dispositif WTB du TCN

3.4.6.1 Dispositif TCN

Un dispositif WTB conforme TCN doit mettre en œuvre au moins un bus MAU pour le WTB.

3.4.6.2 Options WTB

Un MAU WTB doit pouvoir être configuré soit pour un blindage flottant soit pour un blindage à la masse.

Un MAU WTB peut utiliser un connecteur comme spécifié dans la présente norme.

Un MAU WTB peut utiliser un support redondant comme spécifié dans la présente norme.

Un MAU WTB peut mettre en œuvre le nettoyage des contacts comme spécifié dans la présente norme.

Un MAU WTB peut transférer les Données de Processus en Trames Maître comme spécifié dans la présente norme.

3.4.6.3 Options RTP

Un dispositif TCN doit mettre en œuvre des services de Variables.

Sauf pour les dispositifs MVB de Classe 1, un dispositif TCN doit mettre en œuvre les services de Messagerie.

Un dispositif TCN peut mettre en œuvre la fonction de routeur en présence de plusieurs MAU.

Un noeud TCN peut mettre en œuvre le répertoire de nœud.

Un dispositif TCN peut mettre en œuvre le répertoire de stations.

Un dispositif TCN peut mettre en œuvre le protocole de distribution.

3.4.6.4 Options TNM

Un dispositif TCN doit mettre en œuvre la fonction Agent.

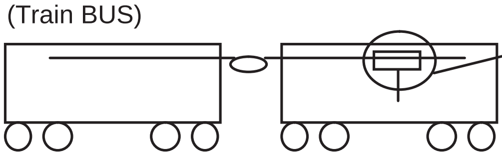
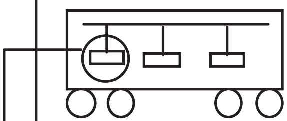
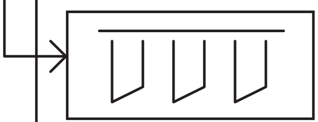
Un dispositif TCN peut mettre en œuvre la fonction Gestionnaire.

3.5 Essai de conformité

Pour être déclarés conformes à la norme TCN, les dispositifs doivent passer avec succès une série d'essais.

Pour s'assurer que l'interopérabilité est garantie conformément au Tableau 5, la présente norme comprend un ensemble de Directives pour l'Essai de Conformité (énumérées dans la CEI 61375-2-2).

Tableau 5 – Essai d'interopérabilité

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

Légende

Anglais	Français
COVERED BY	COUVERT PAR
End USER	UTILISATEUR final
Level 1	Niveau 1
Interface between consists	Interface entre rames
(Train BUS)	(BUS de Train)
USER MESSAGES	MESSAGES UTILISATEUR
MECHANICAL (connectors cables)	MÉCANIQUE (câbles du connecteur)
PROTOCOL PHYSIC/ELEC	PROTOCOLE PHYSIQUE/ELEC
Node connector cable specification (Z: attenuation)	Spécification du câble du connecteur du nœud (Z: atténuation)
Level 2	Niveau 2
Interface between equipment	Interface entre équipements
(Consist Network)	(Réseau de Rames)
Level 3	Niveau 3
Interface between boards or Inter Components	Interface entre cartes ou entre composants
Not Covered	Non Couvert

4 Couche physique

Le présent article spécifie le support physique du WTB comme un bus de paires torsadées et blindées opérant à 1,0 Mbit/s.

La présente norme a pour objet de faire en sorte que les différentes sections et les différents nœuds et connecteurs fournissent un support électrique aussi uniforme que possible, en ce qui concerne la propagation des signaux.

4.1 Topologie

4.1.1 Sections du bus

Le bus WTB doit doit être constitué de nœuds interconnectés par des sections de bus des types suivants:

a) les câbles principaux circulant le long d'une rame (les véhicules de continuité ne possèdent qu'un câble principal);
b) les câbles de jonction reliant les câbles principaux des différentes rames;
c) les câbles d'extension qui prolongent le câble principal jusqu'à un nœud.

4.1.2 Coupleurs

Des connecteurs et des boîtes de jonction peuvent être utilisés pour assembler les nœuds et les sections de câble.

Chaque rame comprend une partie du bus et un certain nombre de nœuds.

4.1.3 Nœuds

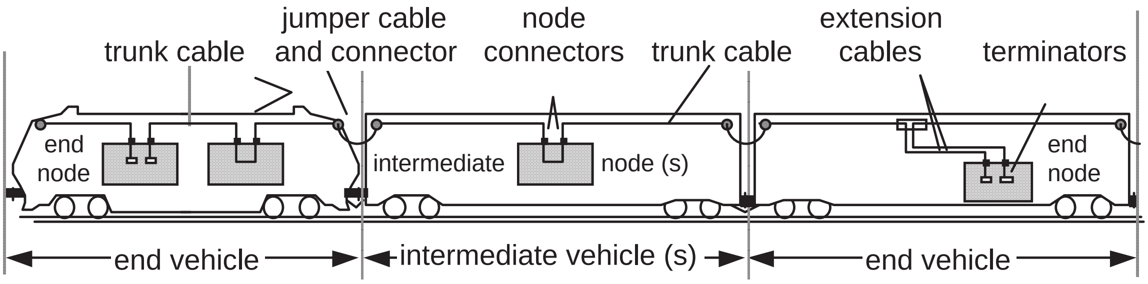
En fonctionnement normal, chaque nœud doit être intégré dans un câble principal et relié à deux sections de bus:

a) les nœuds aux extrémités du bus (Nœuds d'Extrémité) doivent terminer électriquement les deux sections de bus auxquelles ils sont reliés;

b) les nœuds au milieu du bus (Nœuds Intermédiaires) doivent connecter électriquement les deux sections de bus auxquelles ils sont reliés.

Les deux sections de câble auxquelles un nœud est relié doivent s'appeler Direction_1 et Direction_2.

Il peut exister un nœud par rame ou plusieurs nœuds par véhicule d'un train (voir la Figure 9).



Légende

Anglais	Français
trunk cable	câble principal
jumper cable and connector	câble de jonction et connecteur
node connectors	connecteurs de nœud
extension cables	câbles d'extension
terminators	terminaisons
end vehicle	véhicule d'extrémité
intermediate vehicle(s) Intermediate node(s) end node	véhicule(s) intermédiaire(s) Nœud(s) intermédiaire(s) nœud d'extrémité

Figure 9 – Composition du Train (montrant deux Nœuds Intermédiaires)

4.1.4 Orientation de la rame

Lorsque l'orientation des nœuds est critique pour la reconnaissance gauche droite, les conventions suivantes doivent être appliquées:

a) une extrémité de la rame est identifiée comme Extrémité 1, et l'autre comme Extrémité 2;

b) la Direction_1 d'un nœud est reliée à l'Extrémité 1 et la Direction_2 à l'Extrémité 2;

c) si la Direction_1 est orientée vers le nord, le côté de la rame vers l'ouest est appelé côté A, et celui vers l'est côté B;

d) un nœud utilise les mêmes conventions pour A et B que la rame dans laquelle il se trouve.

NOTE 1 La Direction_1 d'un nœud peut être orientée vers la Direction_1 ou la Direction_2 d'un autre nœud, sauf si les nœuds se trouvent dans la même rame, puisque l'orientation d'une rame par rapport à l'autre ne peut être prévue.

NOTE 2 L'Extrémité 1 d'une rame est l'extrémité de la rame opposée à celle où se trouve le frein d'immobilisation.

4.1.5 Spécification de la rame (informelle)

Dans la mesure où un fabricant peut fournir des rames ou des nœuds individuels plutôt qu'un train complet, les spécifications du bus, de la rame et des nœuds sont traitées séparément.

La présente norme spécifie les caractéristiques de l'ensemble du bus et de chaque nœud. Pour une application donnée, les caractéristiques de la rame sont déduites de ces caractéristiques. La conformité du train est alors respectée à condition que le nombre de véhicules, de rames ou de nœuds ne soit pas dépassé.

Les calculs suivants s’appliquent aux véhicules ferroviaires spécifiés dans le CODE UIC 556. D'autres applications, telles que les systèmes de transport urbains, peuvent utiliser un calcul similaire.

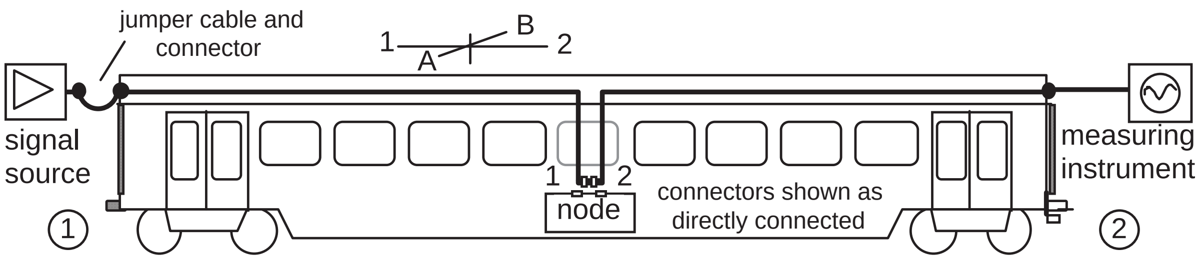
La composition du train de référence dans le CODE UIC 556 comprend 22 véhicules, ce qui donne une longueur de câble de 860,0 m sans recours aux répéteurs. En général, il n'y a qu'un nœud par rame, mais jusqu'à 10 rames composées d’un seul véhicule (une remorque pilote, par exemple) peuvent prendre en charge un deuxième nœud, ce qui donne un total de 32 nœuds.

Selon 4.5.2, un nœud atténue le signal de moins de 0,3 dB (à la fréquence nominale), l'atténuation totale due aux nœuds ne dépasse pas 32 0,3 dB = 9,6 dB.

Selon 4.6.3, les récepteurs sont capables de traiter une plage dynamique de 20,0 dB, l'atténuation restante, pour le câble, les connecteurs et les autres composants, sur l'ensemble du train, ne peut pas dépasser 20,0 – 9,6 = 10,4 dB.

L'atténuation maximale allouée à un véhicule est donc de 10,4 dB/22 = 0,5 dB.

Cette valeur est mesurée avec les nœuds retirés et leurs connexions shuntées. La mesure prend en compte le câble de jonction (voir la Figure 10).



Légende

Anglais	Français
signal source	source du signal
jumper cable and connector	câble de jonction et connecteur
node	nœud
connectors shown as directly connected	connecteurs représentés directement connectés
measuring instrument	instrument de mesure

Figure 10 – Mesure du véhicule

La longueur du câble représente environ 150 % de la longueur du véhicule, à cause des méandres et des câbles d’extension. Pour un véhicule de 26,0 m de long, le support doit couvrir une distance de plus de 860,0 m (22 26,0 1,5 = 858,0 m).

Pour satisfaire à ces exigences, il convient que le support présente une atténuation de moins de 10,4 dB/860,0 m, ou 12,0 dB/km. Dans la mesure où les câbles de jonction, les connecteurs et les épissures peuvent engendrer une atténuation supérieure, un câble principal d'une atténuation de moins de 10,0 dB/km est préconisé (4.2.4).

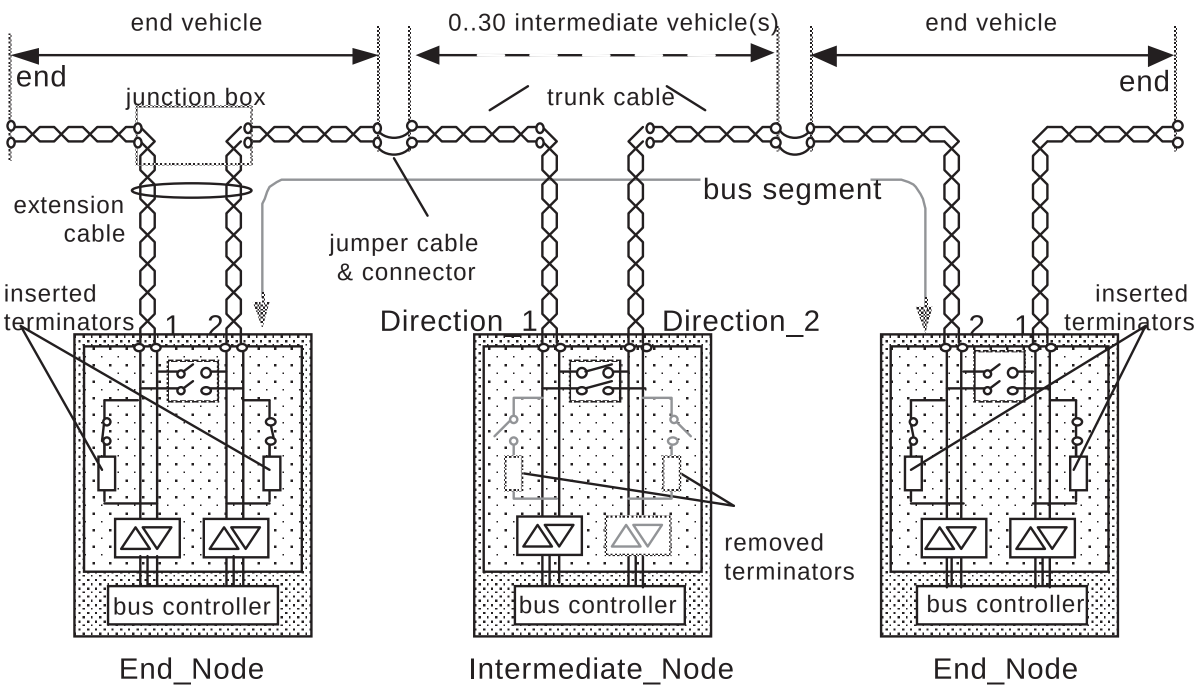
Ce principe s'applique également aux autres paramètres de distorsion. Le système de mesure est détaillé en 4.5.2.1.

Lorsqu'une rame est équipée de Line_A et Line_B redondantes, cet essai s'applique à chaque ligne séparément.

4.2 Spécifications du support

4.2.1 Topologie

Les nœuds doivent être insérés dans le câble WTB, chaque nœud étant relié à deux sections de bus, comme le montre la Figure 11.



Légende

Anglais	Français
end	extrémité
end vehicle	véhicule d'extrémité
0..30 intermediate vehicle(s)	0..30 véhicule(s) intermédiaire(s)
junction box	boîte de jonction
trunk cable	câble principal
extension cable	câble d'extension
jumper cable and connector	câble de jonction et connecteurs
bus segment	segment de bus
inserted terminators	terminaisons insérées
removed terminators	terminaisons retirées
bus controller	contrôleur de bus

Figure 11 – Nœuds reliés en fonctionnement normal

Un nœud doit être capable:

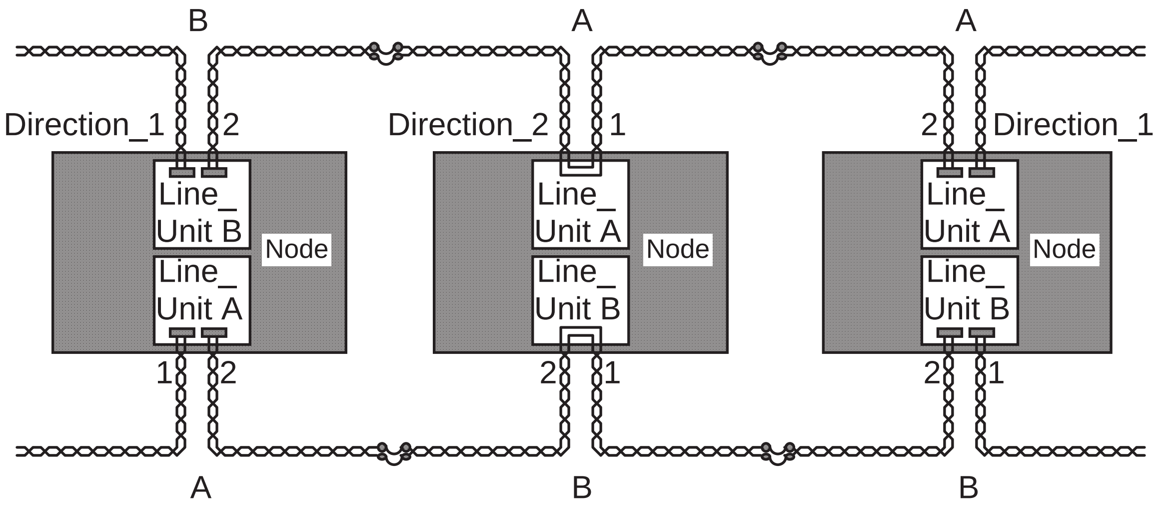
a) d'établir la continuité électrique entre les deux sections de bus qui lui sont reliées, pour assumer la fonction de Nœud Intermédiaire, ou

b) de terminer électriquement les sections de bus reliées par une terminaison (circuit d'adaptation d'impédance), pour assumer la fonction de Nœuds d'Extrémité.

Les Nœuds d'Extrémité doivent pouvoir être capables d'émettre et de recevoir indifféremment sur leur deux sections de bus, tandis que les Nœuds Intermédiaires ne doivent avoir qu'un seul émetteur-récepteur activé.

4.2.2 Support doublé (en option)

La présente norme définit un système de redondance selon lequel chaque nœud est relié à deux lignes par des Unités de Ligne indépendantes (voir la Figure 12).



Légende

Anglais	Français
Node	Nœud

Figure 12 – Liaison à ligne double

Lorsque cette option est utilisée, les spécifications suivantes s'appliquent:

a) les lignes doivent être appelées Line_A et Line_B;

b) cette appellation doit être cohérente pour tous les nœuds à l'intérieur de la même rame;

c) les câbles appartenant aux différentes lignes doivent être marqués distinctement;

d) la configuration de la Line_A et de la Line_B doit être identique par rapport à la Direction_1 et la Direction_2.

NOTE 1 Pour les véhicules UIC, le support à ligne double est obligatoire puisqu'il n'est pas possible de ne relier qu'une seule ligne entre les véhicules.

NOTE 2 La Line_A est associée au côté A de la rame et la Line_B au côté B.

NOTE 3 Etant donné que l'orientation des rames n'est pas prévisible, la Line_A d'une rame peut être connectée à la Line_B d'une autre rame (voir la Figure 12).

4.2.3 Règles de configuration de bus

Ces spécifications s'appliquent à un bus fonctionnant à ses capacités maximales.

Sauf indication contraire, toutes les valeurs électriques doivent être mesurées avec un signal sinusoïdal de 1,0 MHz ± 0,01 %, dont l'amplitude différentielle est de ± 4,0 V (8,0 Vcc).

4.2.3.1 Vitesse de signal

Tous les segments du bus doivent fonctionner à la même vitesse de 1,0 Mbit/s ± 0,01 %, ce qui, en raison du codage Manchester, correspond à une fréquence de signal de 1,0 MHz (BT = 1,0 µs, BR = 1,0 MHz).

4.2.3.2 Retard dû aux nœuds et au câblage

T_pd, le temps de propagation de bout en bout du bus ne doit pas dépasser 60,0 µs entre deux nœuds.

NOTE 1 Le temps de propagation pour une application donnée peut être évalué par

T_pd = (L × 6,0 + R × T_rd)

où:

L est la longueur du câble (câbles principal, d'extension et de jonction) en m;

6,0 ns/m est une valeur approximative du temps de propagation d'une ligne de transmission sous charge;

R représente le nombre de répéteurs; et

T_rd est le temps de propagation d'un répéteur.

NOTE 2 Le WTB peut couvrir jusqu'à 860,0 m sans recours aux répéteurs mais, pour certaines applications, les répéteurs peuvent être utiles.

NOTE 3 Cette spécification est conforme au temps admis selon 5.2.2.2 et 4.7.2.2.

4.2.3.3 Atténuation due aux nœuds et au câblage

L'atténuation de tension totale entre deux nœuds du même segment ne doit pas dépasser 20,0 dB, mesurée avec un signal sinusoïdal à une fréquence comprise entre 0,5 BR et 2,0 BR.

NOTE 1 Cette atténuation est proportionnelle au nombre de nœuds et à la longueur totale du câble.

NOTE 2 Cette spécification correspond à la plage dynamique du récepteur, spécifiée en 4.6.3.

4.2.3.4 Gigue due aux nœuds et au câblage

Un segment correctement terminé, à sa longueur maximale et comportant le nombre maximal de nœuds ne doit pas ajouter plus de ± 0,1 BT de gigue par rapport aux passages au zéro idéaux.

Conditions d'essai:

- la ligne est alimentée par une source d'une amplitude différentielle de 4,0 Vcc ± 10 %, centrée sur 0,0 V, ayant une impédance de source de 22,0 Ω ± 10 %;
- le signal d'attaque est une séquence pseudo aléatoire de symboles Manchester '0' et '1' avec une période de répétition d'au moins 511 bits.

NOTE 1 L'interférence et les réflexions dues aux désadaptations d'impédance entre les sections de bus, les branches, les connecteurs ou les accumulations de charge peuvent provoquer une gigue dans la cadence des passages par zéro.

NOTE 2 Cette spécification, tirée de l'ISO/CEI 8802-3, correspond à la gigue admise du récepteur spécifiée en 4.6.3.

4.2.3.5 Décalage entre les lignes redondantes (en option)

La différence maximale entre le temps de propagation pour la Line_A et la Line_B ne doit pas dépasser T_skew = 30,0 µs entre deux nœuds quelconques.

NOTE Cette spécification correspond au décalage admissible du récepteur, spécifié en 4.7.2.4.1.

4.2.4 Spécification du câble

4.2.4.1 Mécanique

Toutes les sections de câble doivent être composées d’une paire gainée, torsadée et blindée.

La paire doit comporter au moins 12 torsades par mètre.

La surface de section recommandée des fils du câble principal est de 0,75 mm^2 (AWG 18).

La surface de section recommandée des fils du câble de jonction est de 1,34 mm^2 (AWG 16).

Si une connexion indirecte par connecteurs Sub-D est utilisée selon 4.3.4, la surface de section de chaque fil du câble d’extension ne doit pas dépasser 0,56 mm^2 (AWG 20).

4.2.4.2 Marquage

Les fils individuels d'une paire torsadée doivent être appelés X et Y et le blindage appelé S.

Les différents fils du câble doivent être marqués distinctement.

Le marquage doit être maintenu pour tous les points de connexion et de jointure.

4.2.4.3 Impédance caractéristique

Toutes les sections du bus doivent présenter une impédance caractéristique différentielle de Zw = 120,0 Ω (± 10 %) mesurée avec un signal sinusoïdal à une fréquence comprise entre 0,5 BR et 2,0 BR.

4.2.4.4 Atténuation due au câble

Il est recommandé que le câble n’atténue pas le signal de plus de 10,0 dB/km, mesuré avec un signal sinusoïdal à une fréquence 1,0 BR, et de plus de 14 dB/km à 2,0 BR.

4.2.4.5 Capacité répartie du câble

La capacité répartie différentielle (fil à fil) du câble ne doit pas dépasser 65 pF/m à 1,0 BR.

4.2.4.6 Différence de capacité par rapport au blindage

La différence de capacité par rapport au blindage ne doit pas dépasser 1,5 pF/m à 1,0 BR.

4.2.4.7 Rejet de diaphonie

Lorsque le même câble d’extension comporte deux paires de fils, le rejet du signal d'une paire vers l'autre doit excéder 55,0 dB dans la plage comprise entre 0,5 BR et 2,0 BR.

4.2.4.8 Qualité de blindage

L'impédance de transfert du câble doit être inférieure à 20,0 mΩ/m à 20,0 MHz, mesurée selon 5.1.5.1.2 de la CEI 61375-2-2.

L'impédance de transfert différentielle du câble doit être inférieure à 2 mΩ/m, mesurée selon 5.1.5.1.2 de la CEI 61375-2-2.

4.2.4.9 Qualité des connecteurs

NOTE Ces exigences ne s’appliquent pas aux connecteurs entre véhicules.

Toutes les connexions du câble doivent assurer la continuité des fils et du blindage avec une résistance inférieure à 10,0 mΩ.

L'impédance de transfert du connecteur entre une broche et le blindage et entre deux broches, mesurée à 20,0 MHz selon 5.1.5.1.2 de la CEI 61375-2-2, doit être inférieure à 20,0 mΩ et 2,0 mΩ respectivement.

4.2.5 Concept du blindage

Pour se conformer aux différentes applications, deux concepts de blindage sont spécifiés:

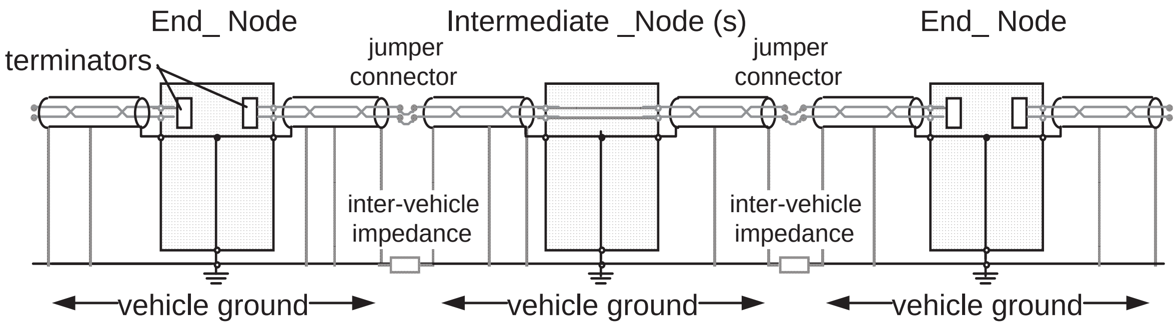
- un concept de blindage à la masse (solution préférée) et
- un concept de blindage flottant.

4.2.5.1 Concept de blindage à la masse

En appliquant le concept du blindage à la masse:

- les blindages doivent être reliés directement à la masse de chaque nœud;
- les câbles de jonction ne doivent pas établir de continuité du blindage entre les véhicules (voir la Figure 13).

NOTE Dans la mesure du possible, il convient de connecter les blindages à la masse (à l'extrémité des véhicules, aux armoires, par exemple) afin de reboucler les courants induits sur des chemins courts et empêcher qu'ils ne génèrent des perturbations CEM à travers le blindage. Cela implique une bonne conductivité de la caisse du véhicule afin d'éviter les courants vagabonds importants au niveau des équipements de traction.



Légende

Anglais	Français
terminators	terminaisons
jumper connector	connecteur de jonction
inter-vehicle impedance	impédance entre véhicules
vehicle ground	masse du véhicule

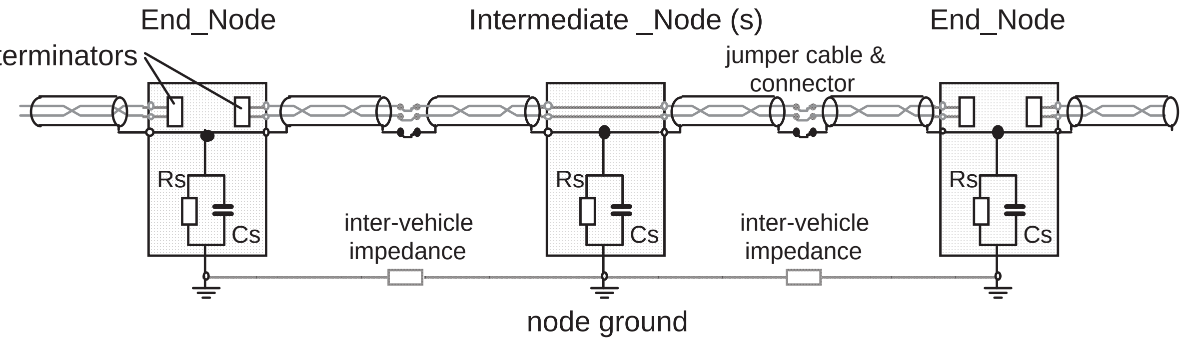
Figure 13 – Concept de blindage à la masse

NOTE Le concept de blindage à la masse est préconisée par la fiche 558 de l'UIC.

4.2.5.2 Concept de blindage flottant

En appliquant le concept de blindage flottant:

- le blindage doit être isolé de la masse s'il n'est pas relié à un nœud;
- le blindage doit être relié pour chaque nœud à la masse du noeud par un circuit RC comprenant une résistance de Rs = 47,0 kΩ ± 5 % montée en parallèle avec un condensateur Cs = 100,0 nF ± 10 %, 750,0 V, comme illustré à la Figure 14.



Légende

Anglais	Français
terminators	terminaisons
inter-vehicle impedance	impédance entre véhicules
node ground	masse du nœud
jumper cable and connector	câble de jonction et connecteur

Figure 14 – Concept de blindage flottant

4.2.6 Terminaison

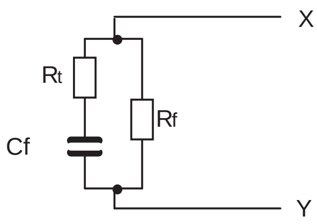
Les Nœuds d'Extrémité doivent terminer électriquement les deux segments de bus auxquels ils sont reliés à l'aide d'une terminaison.

Cette terminaison doit consister en une résistance non polarisée de Zw ± 5 %, avec un angle de phase de moins de 0,087 radians sur une plage de fréquences comprise entre 0,5 BR et 2,0 BR.

La terminaison doit être isolée du blindage du câble.

La terminaison doit présenter une résistance de 2,4 kΩ à un courant continu appliqué entre X et Y capable de dissiper au moins une puissance soutenue de 1,0 W (même pour les applications qui ne nécessitent pas de nettoyage des contacts).

EXEMPLE La Figure 15 illustre un circuit recommandé.



Composants	Type	Valeur
Rt	résistance	130,0 Ω
Rf	résistance	2,4 kΩ
Cf	condensateur	47,0 nF

Figure 15 – Terminaison

4.3 Raccordement au support

Les paragraphes suivants spécifient deux méthodes de connexion d'un nœud:

- les nœuds raccordés directement sont insérés directement dans le câble pincipal sans connecteur;
- les nœuds raccordés indirectement sont reliés au câble principal par des connecteurs.

4.3.1 Identification des points de connexion des nœuds

Un nœud doit identifier les deux sections de bus qui lui sont reliées comme ‘Direction_1’ et ‘Direction_2’, uniquement par rapport à ce nœud.

Un nœud doit identifier les deux lignes qui lui sont reliées comme ‘Line_A’ et ‘Line_B’. Si une seule ligne est utilisée, il doit s’agir de Line_A.

Le raccordement de câble pointe vers l’Unité de Ligne et doit s'appeler:

a) A1X, A1Y et A1S pour Line_A1 (Line_A de Direction_1) et

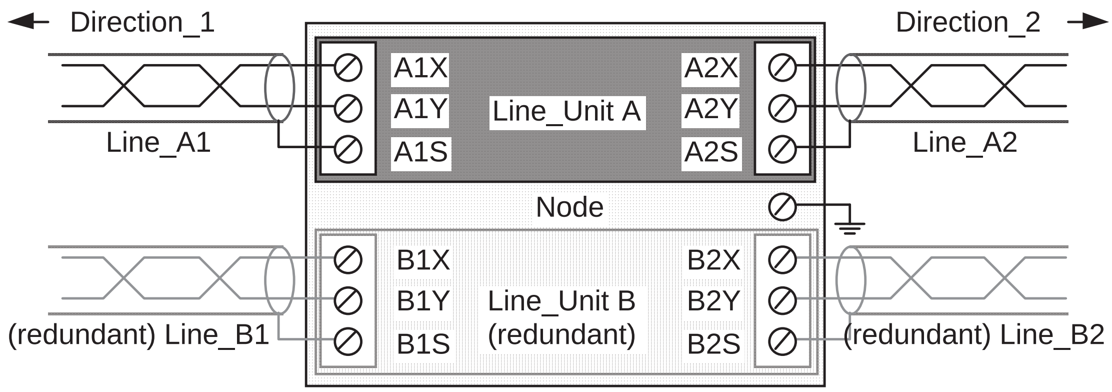
b) A2X, A2Y et A2S pour Line_A2 (Line_A de Direction_2), respectivement

c) B1X, B1Y et B1S pour Line_B1 (Line_B de Direction_1) et

d) B2X, B2Y et B2S pour Line_B2 (Line_B de Direction_2).

4.3.2 Connexion directe d'un nœud

Un nœud à connexion directe doit être inséré dans le câble et fixé au moyen de vis ou d’autres fixations conformes aux exigences électriques et mécaniques (voir la Figure 16).



Légende

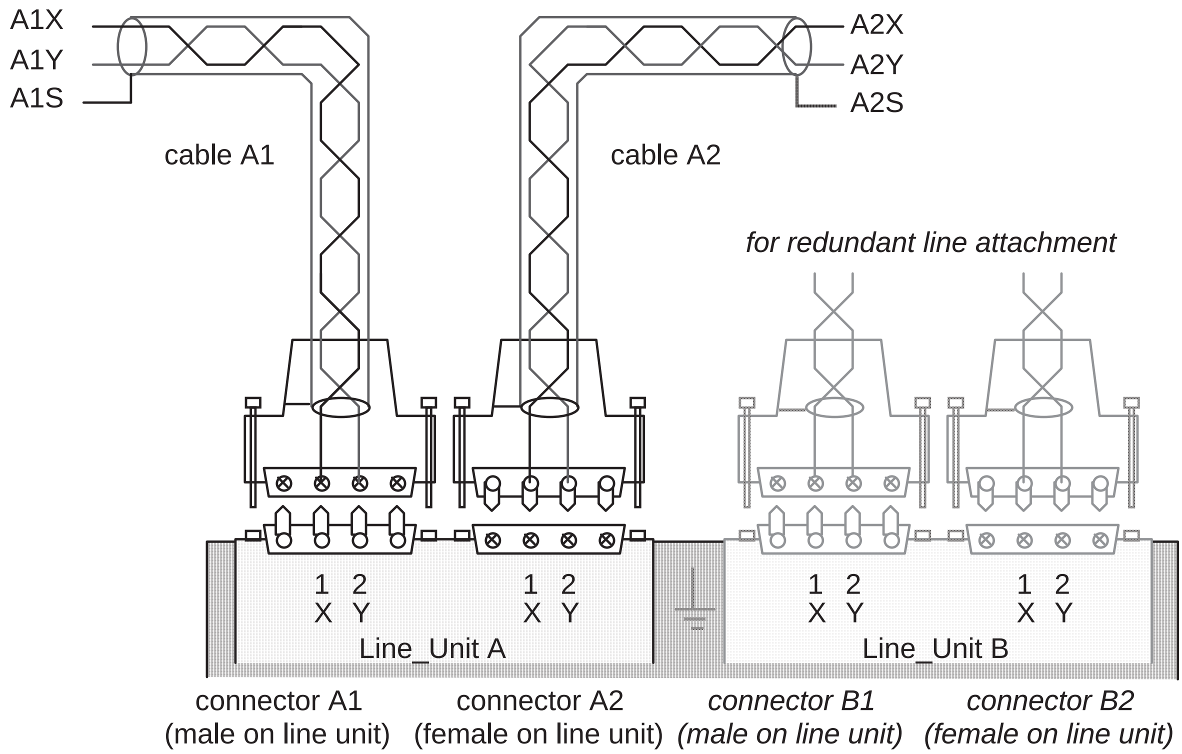
Anglais	Français
(redundant) Line_B1	Line_B1 (redondante)
(redundant) Line_B2	Line_B2 (redondante)
Node	Nœud
Line_Unit B (redundant)	Line_Unit B (redondante)

Figure 16 – Connexion directe d'un nœud (ligne double en option)

Il doit être possible d’enlever le nœud et de relier ensemble les câbles des deux directions pour assurer la continuité du câble et du blindage.

4.3.3 Connexion indirecte d'un nœud

Les nœuds à connexion indirecte doivent utiliser deux connecteurs dans une connexion à une seule ligne, ou quatre connecteurs dans une connexion à deux lignes (voir la Figure 17).



Légende

Anglais	Français
cable A1	câble A1
cable A2	câble A2
for redundant line attachment	pour la connexion de ligne redondante
connector A1 (male on line unit)	connecteur A1 (mâle sur l'unité de ligne)
connector A2 (female on line unit)	connecteur A2 (femelle sur l'unité de ligne)
connector B1 (male on line unit)	connecteur B1 (mâle sur l'unité de ligne)
connector B2 (female on line unit)	connecteur B2 (femelle sur l'unité de ligne)

Figure 17 – Connexion indirecte

4.3.4 Connecteur (en option)

Si l'interchangeabilité est nécessaire, les nœuds à connexion indirecte doivent être reliés au câble de la façon suivante:

a) le connecteur doit être un connecteur D sous-miniature (CEI 60807);

b) le connecteur doit être doté d’un capot blindé conducteur, pour que:

- dans le cas d'un blindage à la masse, le capot soit connecté au blindage du câble pour établir un contact électrique avec le boîtier fixe lors de l’insertion;
- dans le cas d'un blindage flottant, le capot puisse être isolé du blindage du câble;

c) le connecteur doit être équipé de vis métriques;

d) le connecteur doit avoir la disposition et la polarité suivantes:

- La Direction_1 doit utiliser le connecteur mâle sur l’Unité de Ligne et le connecteur femelle sur le câble;
- La Direction_2 doit utiliser le connecteur femelle sur l’Unité de Ligne et le connecteur mâle sur le câble;

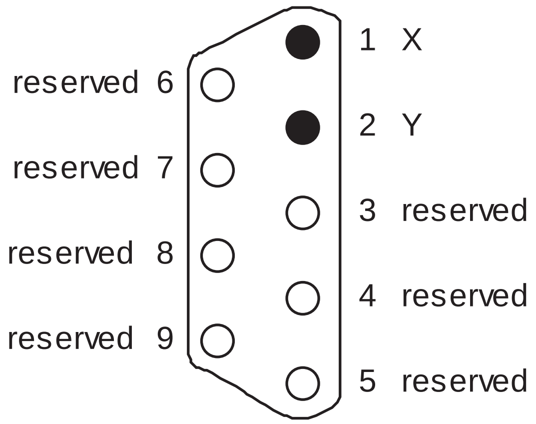
- lorsque les connecteurs d'une même ligne sont disposés verticalement, la Direction_1 doit être le connecteur supérieur, la Direction_2 le connecteur inférieur et Line_A la paire supérieure;
- lorsque les connecteurs d'une même ligne sont disposés horizontalement, la Direction_1 doit être le connecteur de gauche, la Direction_2 le conducteur de droite, en regardant vers le nœud, et Line_A la paire supérieure;

e) il doit être possible de connecter et fixer les connecteurs de câble des deux directions ensemble afin d'assurer la continuité du câble et du blindage;

f) l'affectation des broches du connecteur (mâle ou femelle) doit être conforme à celle spécifiée dans le Tableau 6 et présentée dans la Figure 18.

Tableau 6 – Affectation des broches d'un connecteur WTB

1	X fil positif	6	réservé
2	Y fil négatif	7	réservé
3	réservé pour le blindage	8	réservé
4	réservé	9	réservé
5	réservé		



Légende

Anglais	Français
reserved	réservé

Figure 18 – Face avant d'un connecteur WTB

NOTE Il est préférable pour le blindage flottant de séparer électriquement le réceptacle par rapport au boîtier du nœud.

4.4 Spécifications des nœuds

4.4.1 Différents éléments d'un nœud

Un nœud doit être relié au support par son Unité de Connexion au Support (MAU).

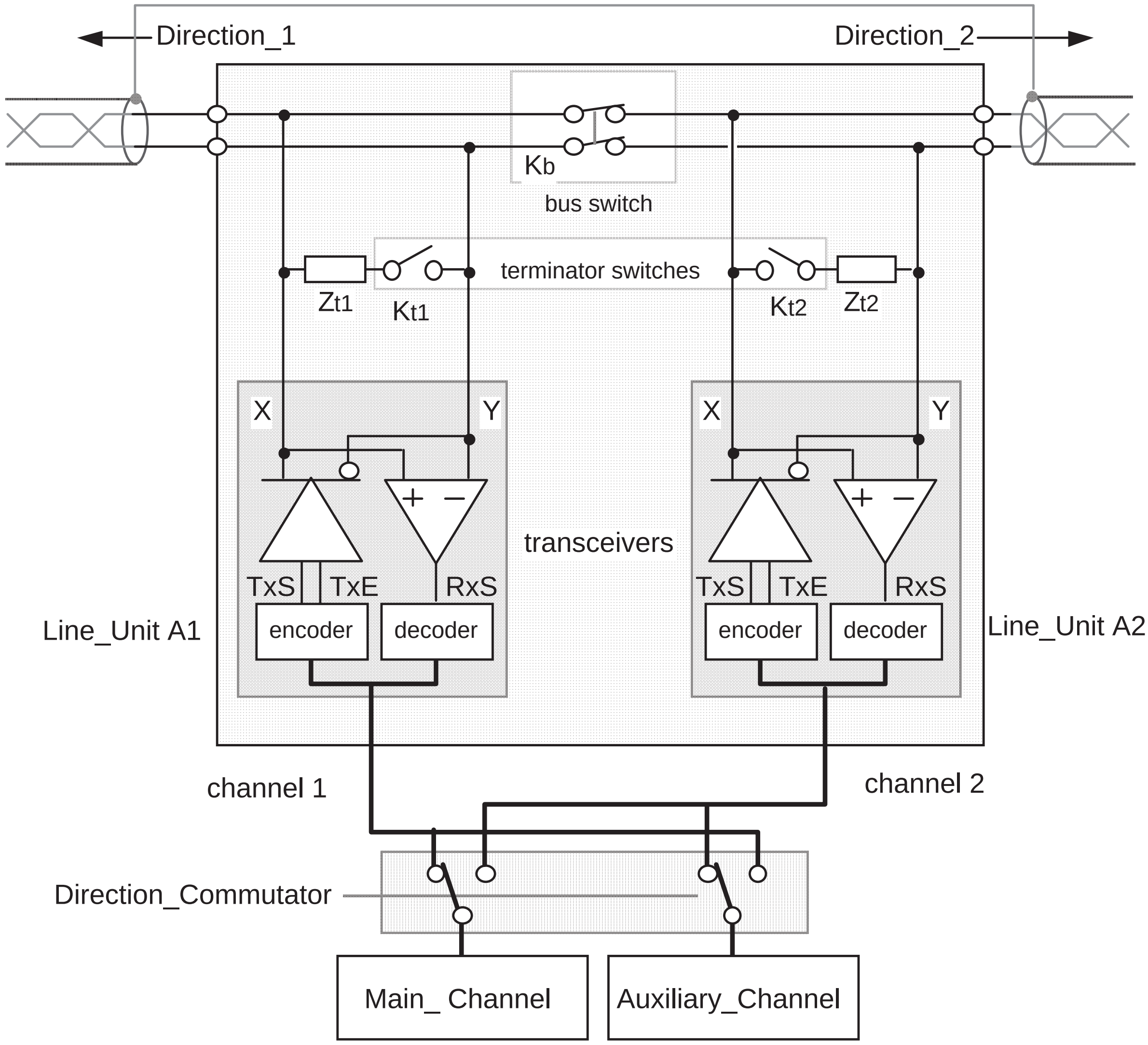
Pour une connexion à ligne simple, la MAU doit comprendre:

a) une Unité de Ligne;

b) un Commutateur de Direction;

c) un Canal Principal et un Canal Auxiliaire.

EXEMPLE La Figure 19 présente une MAU dont les commutateurs sont à la position Intermediate_Setting.



Légende

Anglais	Français
bus switch	commutateur de bus
terminator switches	commutateurs de terminaison
transceivers	émetteurs-récepteurs
encodeur	codeur
channel	canal

Figure 19 – Exemple de structure d'une MAU

4.4.1.1 Composants d'une Unité de Ligne

Une Unité de Ligne doit comprendre:

a) un commutateur de bus (Kb) qui établit ou interrompt la connexion dans les deux directions;

b) un commutateur de terminaison pour chaque direction (Kt1, Kt2) qui insère une terminaison (Zt1, Zt2) dans la position End_Setting ou l'enlève dans la position Intermediate_Setting;

c) deux circuits émetteur-récepteur, un pour chaque direction. Chaque émetteur est commandé par les signaux binaires TxS (signal) et TxE (permission). La sortie du récepteur est RxS (un signal numérique ou analogique). Les émetteurs-récepteurs sont isolés galvaniquement par rapport à la ligne par un dispositif approprié (un transformateur, par exemple);

d) deux codeurs/décodeurs Manchester, un pour chaque émetteur-récepteur, qui peuvent être intégrés dans leur émetteur ou récepteur respectif. La sortie et l'entrée du codeur/décodeur Manchester sont identifiées comme une interface modem;

e) des circuits de protection contre les surtensions et les courts-circuits qui sont reliés à des contacteurs d'isolement (non montrés dans la Figure 19).

NOTE 1 Un nœud relié à un support redondant possède deux Unités de Ligne.

NOTE 2 Les commutateurs Kb et Kt1 ou Kt2 peuvent être des contacts du même relais mécanique, dans le cas où ce type de relais est utilisé.

4.4.1.2 Canal Principal et Canal Auxiliaire

Le Canal Principal et le Canal Auxiliaire doivent être capables d'émettre et de recevoir les trames HDLC et les signaux de commande en provenance de et vers les Unités de Ligne.

NOTE Puisque le Canal Auxiliaire émet et reçoit uniquement les trames afin de détecter les nœuds supplémentaires et recevoir son adresse, son fonctionnement peut être simplifié par rapport à celui du Canal Principal.

4.4.1.3 Commutateur de Direction

Le Commutateur de Direction doit relier le Canal Principal à la Direction_1 et le Canal Auxiliaire à la Direction_2, ou inversement.

NOTE Le Commutateur de Direction n'est pas forcément un élément physique. Il peut être mis en œuvre dans la logique ou le logiciel.

4.4.2 Position du nœud et des commutateurs

Le présent paragraphe définit les caractéristiques du nœud pour les deux positions suivantes: Intermediate_Setting ou End_Setting.

4.4.2.1 Intermediate_Setting

Dans la position Intermediate_Setting, un nœud doit:

a) établir la continuité entre les deux segments de ligne (Kb fermé);

b) enlever les Terminaisons Zt1 et Zt2 (Kt1 et Kt2 sont ouverts);

c) relier le Canal Principal à la ligne en passant par l'émetteur-récepteur 1 ou 2;

d) déconnecter le Canal Auxiliaire et l'émetteur-récepteur non utilisé.

NOTE La position Intermediate_Setting est adoptée par un nœud non fonctionnel (désactivé ou mis hors tension) et par les nœuds qui ne se trouvent pas à l'extrémité du bus.

4.4.2.2 End_Setting

Dans la position End_Setting, un nœud doit:

a) isoler les deux segments (Kb est ouvert);

b) insérer les deux Terminaisons (Kt1 et Kt2 sont fermés);

c) relier le Canal Auxiliaire dans une direction et le Canal Principal dans l'autre direction.

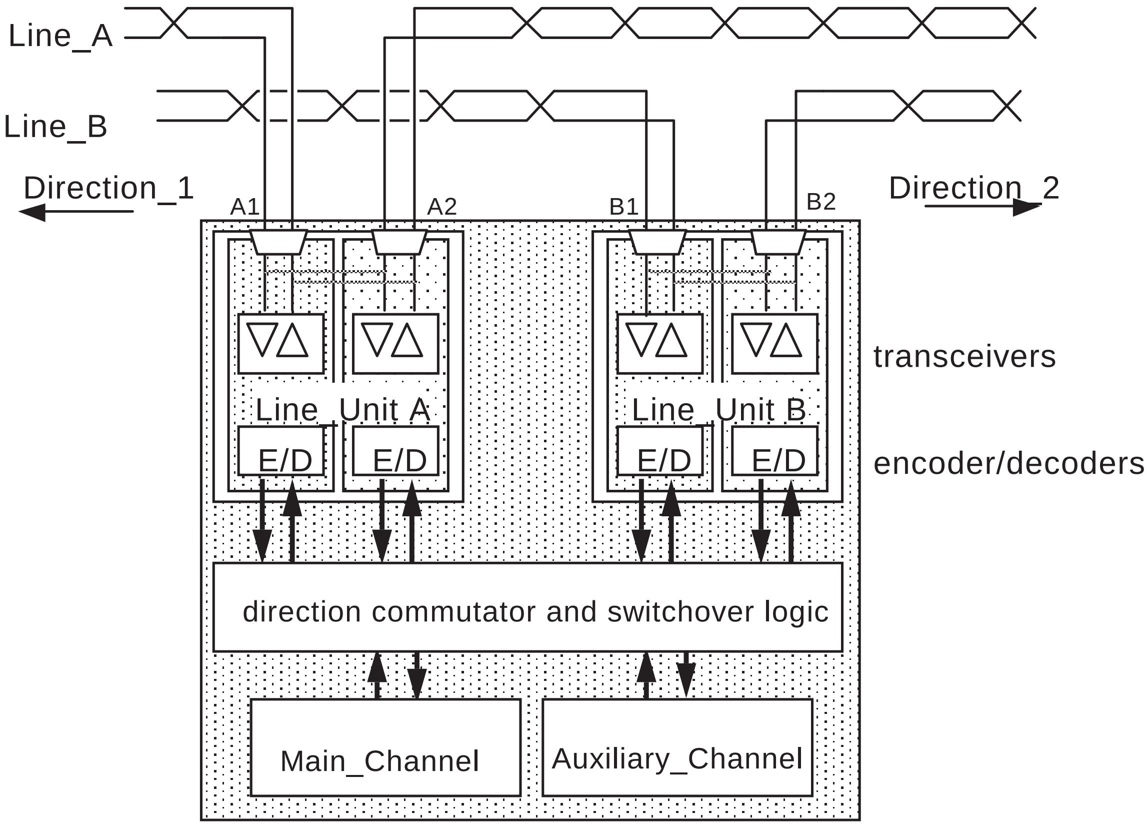
NOTE La position End_Setting est adoptée par un nœud sans nom, un nœud à l'extrémité du bus (et en particulier, un maître sans esclave) ou par un nœud en mode de veille.

4.4.3 Unités de Ligne doublées (en option)

Lorsque cette option est utilisée, les spécifications suivantes s'appliquent:

- les MAU affectées à la connexion doublée doivent être reliées à la Line_A et à la Line_B par des Unités de Ligne différentes;
- la position du nœud (End_Setting ou Intermediate_Setting) doit s'appliquer aux deux Unités de Ligne de la même façon;
- il doit être possible de retirer une ligne tout en maintenant l'autre ligne en fonctionnement.

EXEMPLE La Figure 20 montre une MAU pour l'exploitation de lignes redondantes. La logique de commutation permet la réception de signaux de la Line_A ou de la Line_B.



Légende

Anglais	Français
transceivers	émetteurs-récepteurs
encoder/decoders	codeur/décodeurs
direction commutator and switchover logic	commutateur de direction et logique de commutation

Figure 20 – Nœuds avec des Unités de Ligne redondantes

4.5 Spécifications des Unités de Ligne

Bien que seule l'Unité de Ligne A soit citée, ces spécifications s'appliquent également à l'Unité de Ligne B dans le cas où un support doublé est utilisé.

4.5.1 Isolement galvanique

La tension d'isolement et la résistance d'isolement, entre le boîtier du nœud et l'un des points, A1X, A1Y, A2X ou A2Y, doivent dépasser la valeur spécifiée par la CEI 60571.

NOTE Dans l'édition actuelle, ces valeurs sont de 0,50 kVeff et 1,0 MΩ.

4.5.2 Pertes d'insertion d'une Unité de Ligne

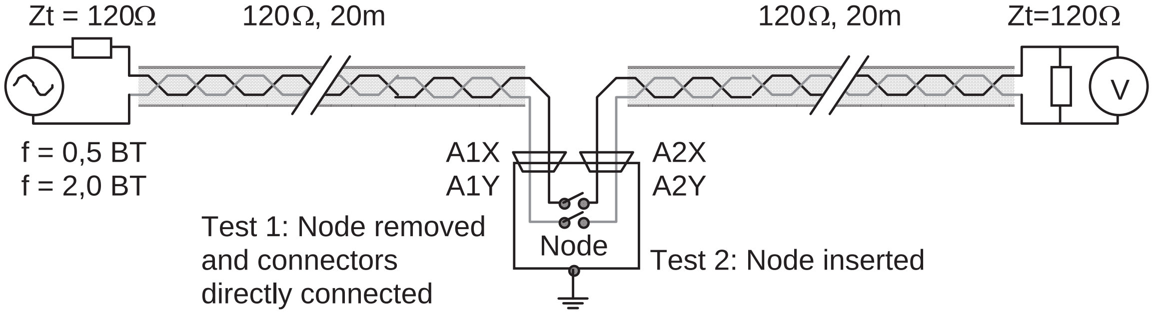
4.5.2.1 Mesure d'atténuation

Pour mesurer la perte d'insertion, le signal sinusoïdal d'un générateur (d'impédance interne = Zt) est appliqué sur les points A1X et A1Y par l'intermédiaire d'un câble de 20,0 m. Elle est mesurée à l'aide d'un voltmètre (relié en parallèle avec une impédance Zt) à l'extrémité d'autres câbles de 20,0 m reliés aux points A2X et A2Y (ou inversement) comme représenté à la Figure 21.

L'atténuation est définie comme le rapport, exprimé en décibels, de deux tensions différentielles:

a) la première tension étant réglée sur 4,0 Vcc lorsque le nœud est retiré et le connecteur du câble branché;

b) la deuxième tension étant mesurée lors de l'insertion du nœud (dans la position End_Setting ou Intermediate_Setting, selon l'essai).



LégendeLégende

Anglais	Français
Test 1: Node removed and connectors directly connected	Essai 1: Nœud retiré et connecteurs directement connectés
Test 2: Node inserted	Essai 2: Nœud inséré
Node	Nœud

Figure 21 – Mesure de l'atténuation

4.5.2.2 Nœud dans la position End_Setting

Une Unité de Ligne dans la position End_Setting (Kb ouvert, Kt1 et Kt2 fermés) doit présenter à un signal de 1,0 BR appliqué entre les points A1X et A1Y ou entre les points A2X et A2Y, une impédance correspondant à la terminaison spécifiée en 4.2.6.

Une Unité de Ligne dans la position End_Setting doit atténuer de plus de 55,0 dB un signal appliqué entre A1X et A1Y et mesuré entre A2X et A2Y (ou inversement).

4.5.2.3 Nœud dans la position Intermediate_Setting

Une Unité de Ligne dans la position Intermediate_Setting (Kb fermé, Kt1 et Kt2 ouverts):

a) avec son récepteur fonctionnant normalement et son émetteur à l'état haute impédance, ou

b) avec le récepteur ou l'émetteur non alimenté,

doit atténuer un signal sinusoïdal:

- de moins de 0,3 dB entre 0,5 BR et 1,0 BR; et

- de moins de 0,4 dB jusqu'à 2,0 BR.

Le nœud doit présenter une résistance d'au moins 1 MΩ par rapport à une tension continue positive ou négative de 48,0 V appliquée entre A1X et A1Y ou entre A2X et A2Y.

4.5.3 Spécifications des commutateurs

Tous les commutateurs reliés à la ligne du bus (Kb, Kt, etc.):

a) doivent présenter un isolement d'au moins 500,0 Veff en position ouverte;

b) doivent présenter une résistance de contact initiale de moins de 0,050 Ω en position fermée;

c) doivent être spécifiés pour une résistance de contact de moins de 0,100 Ω en position fermée après 10^7 cycles;

d) doivent commuter d'une position vers une autre en moins de 10,0 ms, y compris le temps de rebondissement.

NOTE Le relais peut être à semi-conducteurs ou de type mécanique.

4.5.4 Connexions du blindage à une Unité de Ligne

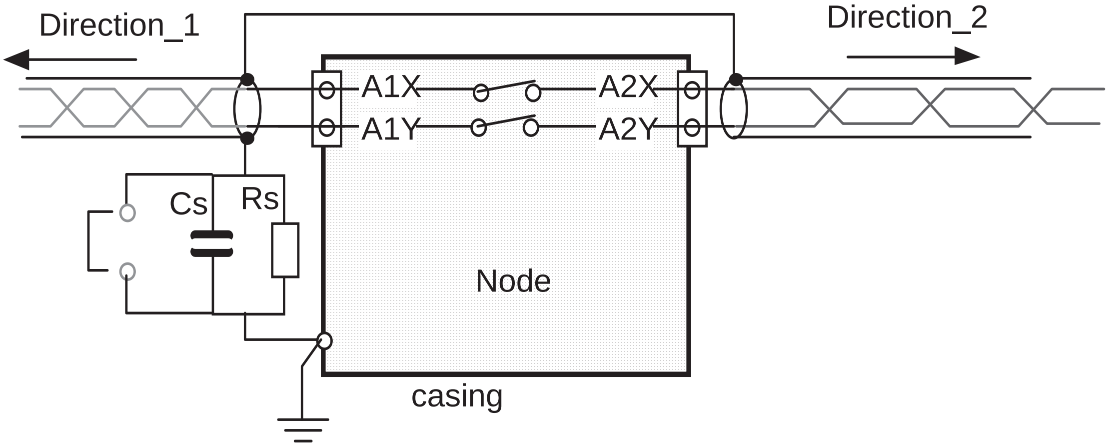
Sur chaque nœud, les blindages des câbles de la Direction_1 et la Direction_2 doivent être reliés directement entre eux à travers les réceptacles avec une résistance de contact de moins de 0,010 Ω.

Une Unité de Ligne doit permettre de connecter les blindages au boîtier du nœud:

a) directement par une basse impédance, comme spécifié en 4.2.5.1 (blindage à la masse);

b) par le réseau RC spécifié en 4.2.5.2 (blindage flottant).

EXEMPLE Le principe de la connexion du blindage dans un nœud est présenté dans la Figure 22.



Légende

Anglais	Français
Node	Nœud
Casing	Boîtier

Figure 22 – Mise à la masse du blindage de l'Unité de Ligne

4.5.5 Nettoyage des contacts (en option)

Pour combattre l'oxydation des contacts des relais et des connecteurs, un nœud peut avoir recours au nettoyage des contacts. Il s'agit d'appliquer une tension continue entre les fils X et Y dans chaque direction.

Si cette option de nettoyage des contacts est utilisée, les spécifications du présent paragraphe s'appliquent.

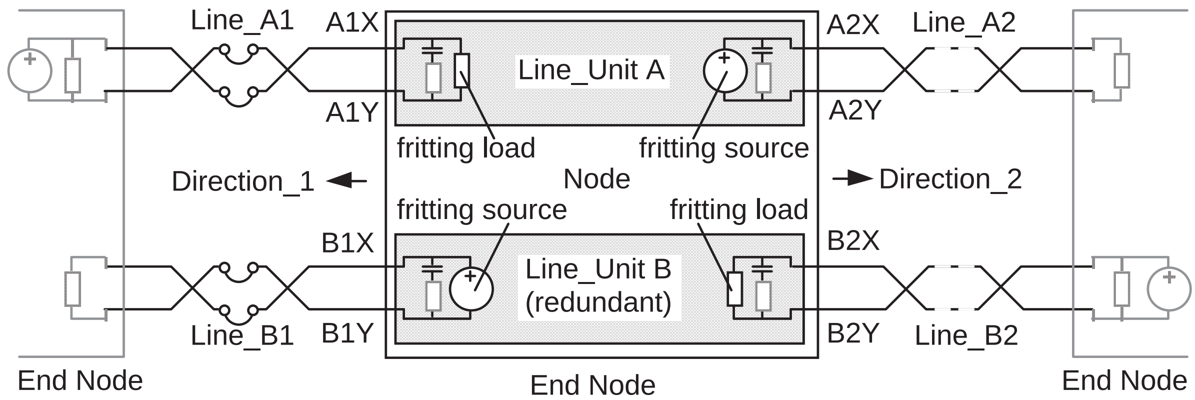
NOTE 1 L'utilisation du nettoyage des contacts n'est pas spécifiée pour un support non redondant.

NOTE 2 Les nœuds qui n'utilisent pas le nettoyage des contacts et ceux qui l'utilisent peuvent être combinés sur le même bus.

4.5.5.1 Source et charge de nettoyage des contacts

Un nœud qui prend en charge le nettoyage des contacts doit fournir une source de tension de nettoyage des contacts et une charge de tension de nettoyage des contacts pour chaque direction.

Dans le cas d'un support physique redondant, un nœud doit fournir deux sources de tension de nettoyage des contacts indépendantes, une pour chaque direction. Il doit également fournir les charges pour la tension de nettoyage des contacts d'un autre nœud (voir la Figure 23).



Légende

Anglais	Français
fritting load	charge de nettoyage des contacts
fritting source	source de nettoyage des contacts
Node	Nœud
Line_Unit B (redundant)	Line_Unit B (redondant)
End Node	Nœud d'extrémité

Figure 23 – Source et charge de nettoyage des contacts

Le pôle positif de la source de nettoyage des contacts doit être relié à A2X et B1X respectivement.

Le pôle négatif de la source de nettoyage des contacts doit être relié à A2Y et B1Y respectivement.

La source de nettoyage des contacts doit fournir une tension continue de 48,0 V +20 %/–10 % mesurée au niveau des points de connexion (A2X/A2Y et B1X/B1Y respectivement)

L'ondulation de la source de nettoyage des contacts ne doit pas dépasser 0,100 Vcc dans la plage de 0,5 BR à 2,0 BR.

Le courant fourni par la source de nettoyage des contacts ne doit pas dépasser 80,0 mA, c.c.

L'isolation entrée/sortie de la source de nettoyage des contacts doit être conforme à la CEI 60571.

La source de nettoyage des contacts doit être découplée de la ligne (par une inductance de 0,10 H ou tout autre dispositif qui respecte les pertes d'insertion d'un nœud, par exemple).

La constante de temps d'enclenchement de la source doit être comprise entre 0,5 ms et 5,0 ms.

La constante de temps de déclenchement de la source doit être comprise entre 0,5 ms et 5,0 ms.

L'atténuation entre les deux sources de tension de nettoyage des contacts du même nœud doit être supérieure à 50,0 dB entre 0,5 BR et 2,0 BR.

4.5.5.2 Mise en œuvre du nettoyage des contacts

Un Nœud d'Extrémité doit enclencher la source de nettoyage des contacts sur son Canal Auxiliaire (un nœud non nommé possède deux Canaux Auxiliaires, un dans chaque direction, alors qu'un nœud en mode veille n'a pas de Canal Auxiliaire).

NOTE 1 Un nœud peut enclencher et déclencher la source de nettoyage des contacts périodiquement (pour diminuer la consommation, par exemple).

NOTE 2 Il est permis d'enclencher la source de nettoyage des contacts sur le Canal Principal si les niveaux de perturbation électromagnétique sont respectés.

4.6 Spécifications de l'émetteur-récepteur

Une MAU reliée à un bus redondant possède quatre émetteurs-récepteurs, appelés A1, A2, B1 et B2. Dans un schéma non redondant, seuls les émetteurs-récepteurs A1 et A2 sont utilisés. Les spécifications suivantes s'appliquent à tous.

4.6.1 Conventions

Sauf indication contraire, les conditions de mesure par défaut suivantes s'appliquent:

a) les caractéristiques d'un émetteur-récepteur sont mesurées aux points X et Y, où les sections du bus sont reliées au nœud;
b) toutes les tensions sont mesurées comme différence de tension entre X et Y (Ux – Uy);
c) lors de la mesure d'un émetteur, le circuit du récepteur est en état normal de réception, en mesurant un récepteur, le circuit de l'émetteur est dans un état d'impédance élevée;
d) toutes les valeurs des résistances sont exprimées ± 1 % et toutes les valeurs des condensateurs sont exprimées ± 10 %.

4.6.2 Émetteur

4.6.2.1 Charge de l'émetteur

Pour définir les caractéristiques de l'émetteur, quatre circuits d'essai sont spécifiés afin de simuler les câbles et les nœuds:

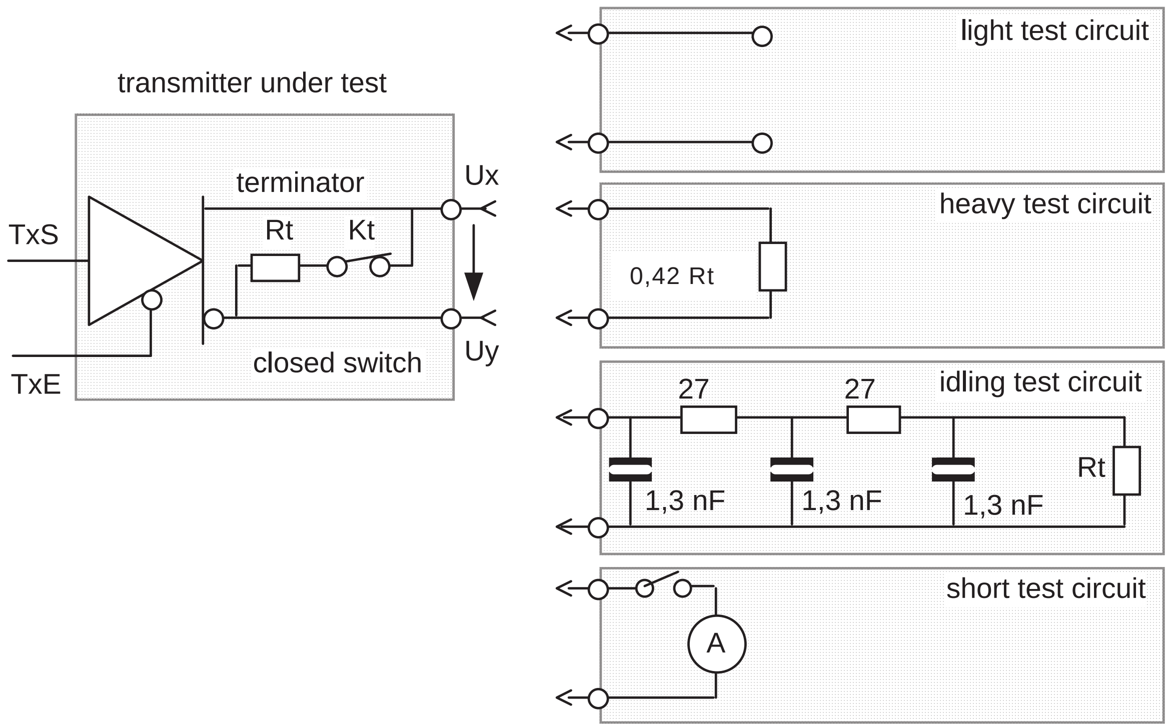
a) le circuit d'essai léger simule une ligne ouverte (comme pour un nœud dans la position End_Setting). La valeur de la charge résistive totale est égale à celle de la terminaison;

b) le circuit d’essai lourd simule un bus entièrement chargé. La valeur de la charge résistive totale est égale à 0,42 de celle de la terminaison;

c) le circuit d'essai au repos simule un câble d'une longueur de 860,0 m sans charge résistive. La valeur de chaque condensateur est de 1,3 nF ± 10 %, celle de chaque résistance étant de 27,0 Ω ± 1 %;

d) le circuit d'essai de court-circuit simule une défaillance de la ligne. Il ne comprend qu'un circuit de mesure de courant;

Ces circuits sont présentés à la Figure 24.



Légende

Anglais	Français
light test circuit	circuit d'essai léger
heavy test circuit	circuit d'essai lourd
transmitter under test	émetteur en essai
terminator	terminaison
closed switch	commutateur fermé
idling test circuit	circuit d’essai à vide
short test circuit	circuit d'essai de court-circuit

Figure 24 – Montages de l’émetteur

e) les mesures sont faites avec le nœud en position End_Setting (Kb ouvert, Kt fermé);

f) la terminaison de l’unité de ligne est prise en compte dans la spécification du circuit d’essai.

4.6.2.2 Signal de sortie de l'émetteur

NOTE Compte tenu du codage de données utilisé, l'émetteur génère des impulsions dont la longueur est d'un bit (1,0 BT) ou d'un demi-bit (0,5 BT) entre le Préambule et le Délimiteur de Fin.

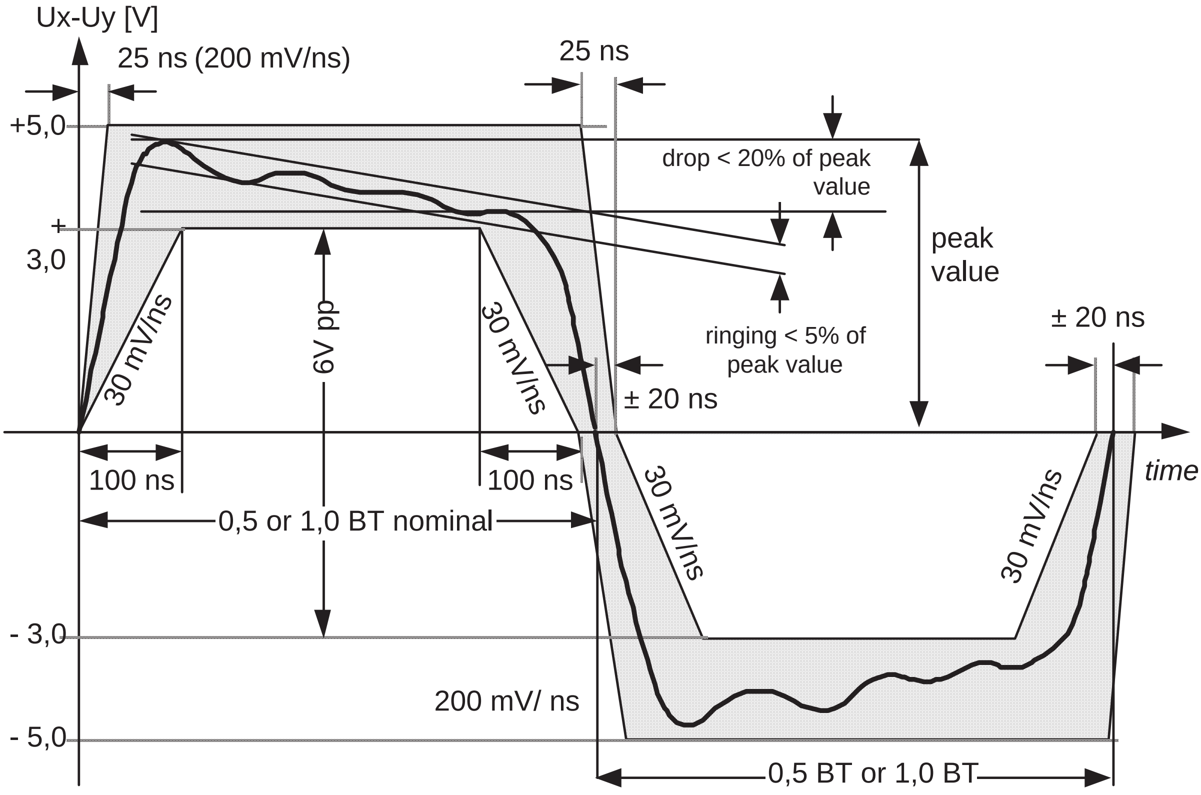
Ces spécifications s'appliquent aux impulsions positives ou négatives de 0,5 BT et de 1,0 BT.

L’émetteur doit avoir une sortie de type différentiel.

Le signal de sortie est la tension différentielle (Ux – Uy) au point de connexion d'un nœud.

Lorsque l'émetteur est relié aux circuits d'essai lourd et léger, définis en 4.6.2.1, il doit se conformer aux spécifications suivantes, selon la Figure 25:

a) le signal de sortie doit être positif, puis négatif en alternance;
b) l'amplitude du signal de sortie doit être d'au moins ± 3,0 V sur le circuit d'essai lourd et de ± 7,0 V au maximum sur le circuit d'essai léger;
c) l'amplitude de crête est définie comme l'amplitude maximale du signal de sortie. L'amplitude du signal ne doit pas diminuer de plus de 20 % par rapport à cette valeur maximale jusqu'à 0,100 µs avant la prochaine transition zéro attendue. Pendant ce temps, les oscillations de l'amplitude par rapport à la chute de tension moyenne ne doivent pas dépasser 5 % de la valeur de crête;
d) la vitesse de variation du signal de sortie doit être inférieure à 0,20 V/ns à tout moment, et supérieure à 0,03 V/ns dans les 100,0 ns du passage par zéro;
e) le dépassement du signal de sortie, défini comme le rapport entre l'amplitude maximale et l'amplitude stationnaire, ne doit pas dépasser 10 % de son amplitude stationnaire;
f) la distorsion frontale du signal de sortie, définie comme la différence de temps entre le passage à zéro théorique et le passage à zéro réel, ne doit pas dépasser ± 2 % d'un Temps Bit.



Légende

Anglais	Français
drop < 20 % of peak value	baisse < 20 % de la valeur crête
peak value	valeur crête
ringing < 5 % of peak value	oscillation < 5 % de la valeur crête
time	durée
0,5 BT or 1,0 BT	0,5 BT ou 1,0 BT
0,5 BT or 1,0 BT nominal	0,5 BT ou 1,0 BT nominal

Figure 25 – Forme d'impulsion au niveau de l'émetteur

NOTE Une chute de tension est prévisible puisque le condensateur de nettoyage des contacts est relié en série au transformateur.

4.6.2.3 Bruit de l'émetteur

Tout bruit généré par un émetteur qui n’est pas en train d’émettre ne doit pas dépasser une valeur de 5 mVeff sur une plage de fréquences comprise entre 1,0 kHz à 4,0 BR.

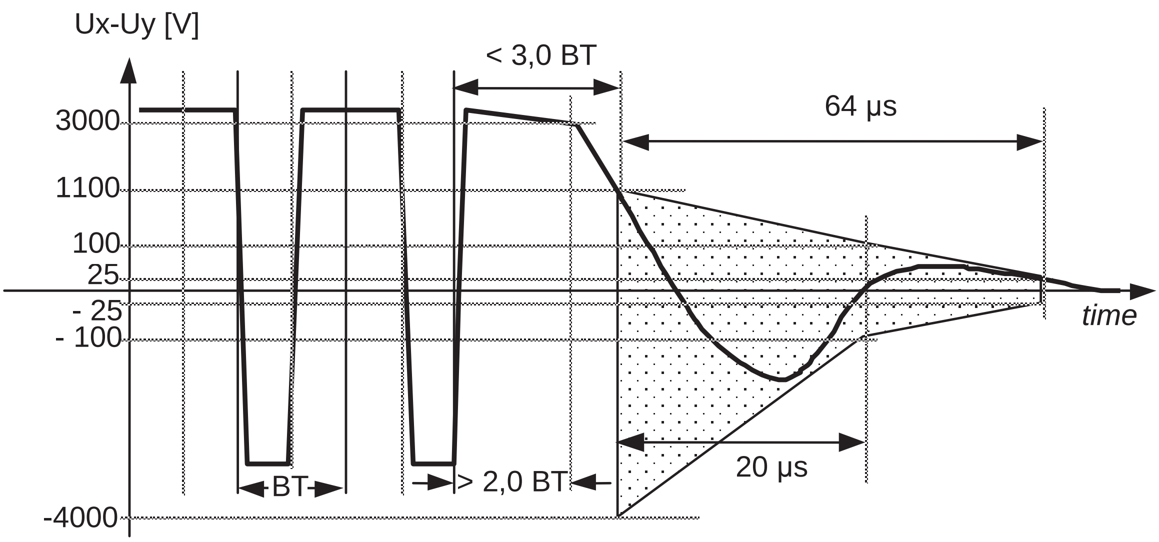
4.6.2.4 Fin de trame de l'émetteur

La fin de trame générée par l’émetteur doit être soumise à essai dans les conditions suivantes:

a) l’émetteur émet la trame la plus longue possible;
b) les bits de trame Frame_Data sont une séquence pseudo aléatoire de symboles ‘1’ et ‘0’;
c) la trame est terminée par le symbole Délimiteur de Fin (End_Delimiter) (voir 4.7.1.4);
d) l’émetteur alimente le circuit d’essai à vide mentionné en 4.6.2.1;
e) l'amplitude différentielle moyenne est supérieure à 4,5 V avant la mise hors tension de l'émetteur.

Dans ces conditions, le signal de sortie doit rester dans les limites suivantes (voir la Figure 26):

1) 100,0 ns après la dernière transition négative à positive et pendant 2,0 BT ± 100 ns, le signal de sortie doit rester au-dessus de 0,300 V;
2) au plus tard 3,0 BT après la dernière transition négative à positive, le signal de sortie doit tomber en dessous de 1,100 V;
3) au plus tard 20,0 µs après que le signal de sortie a atteint 1,100 V pour la première fois, l'amplitude du signal de sortie ne doit pas dépasser 0,100 V;
4) au plus tard 64 µs après que le signal de sortie a atteint 1,100 V pour la première fois, l'amplitude du signal de sortie ne doit pas dépasser 0,025 V.



Légende

Anglais	Français
time	durée

Figure 26 – Signal et mise en veille de l'émetteur

NOTE L'oscillation sur la ligne après la mise en veille de l'émetteur peut être limitée en équilibrant le signal dans chaque cellule de bit. Elle peut être réduite encore en équilibrant le Délimiteur de Fin (End_Delimiter) comme spécifié en 4.7.1.4.

4.6.2.5 Tolérance aux défaillances de l'émetteur

Un émetteur, qu'il soit actif ou non, doit tolérer la mise en place du circuit d'essai de court-circuit (4.6.2.1) au point de connexion tant que la stabilité thermique n'est pas atteinte, et doit reprendre un fonctionnement normal après suppression du circuit d'essai de court-circuit.

Le courant de court-circuit ne doit pas dépasser 1,0 A.

NOTE Pour les essais de conformité, il est considéré que la stabilité thermique est atteinte après 1 h.

4.6.2.6 Dispositif anti-bavardage de l'émetteur

Chaque émetteur doit comprendre un circuit indépendant qui permet de découpler l'émetteur de la ligne de bus si la durée de l'émission dépasse la valeur T_jabber égale à la durée de la trame la plus longue (y compris le Préambule, le Délimiteur de Fin et le bourrage de bits) + 20 %.

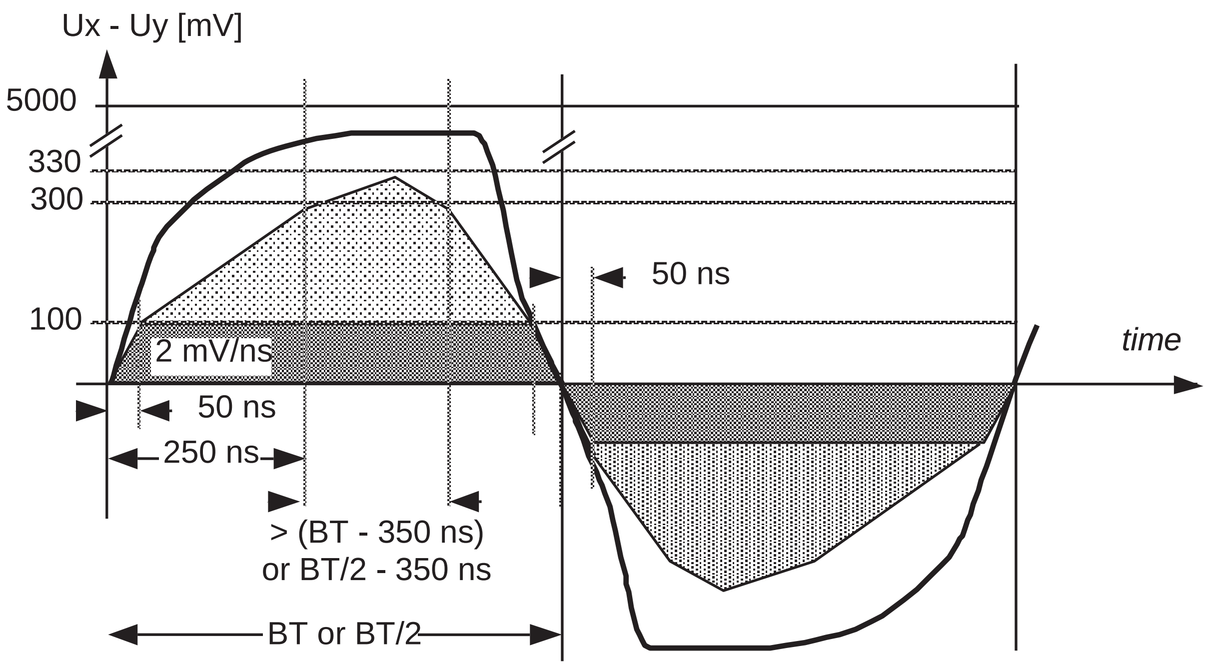
4.6.3 Spécifications du récepteur

4.6.3.1 Caractéristiques du signal du récepteur

Il convient que le récepteur décode un signal de forme suivante (voir la Figure 27), appliqué au point de connexion:

a) lorsque l'amplitude du signal reçu est inférieure à 0,100 V, sa pente est supérieure à 2,0 mV/ns;

b) lorsque le signal reçu reste au-dessus de 0,300 V pendant une période qui commence 100,0 ns après le passage par zéro précédant et qui dure au moins (0,5 BT – 350,0 µs) et (1,0 BT – 350,0 µs) respectivement, tandis que son amplitude de crête varie entre 0,330 V et 5,00 V.



Légende

Anglais	Français
BT or BT/2	BT ou BT/2
time	durée

Figure 27 – Enveloppe du signal du récepteur

Une erreur de trame est détectée comme une trame manquante ou non valable (voir 4.7.1.5.3), une taille de trame incorrecte ou des bits de trame Frame_Data incorrects (erreur FCS).

4.6.3.2 Polarité du récepteur

Un niveau HAUT (HIGH) sur TxS doit correspondre à une tension différentielle positive (Ux – Uy) qui à son tour doit correspondre à un niveau HAUT du signal RxS d'un récepteur.

Un niveau BAS (LOW) sur TxS doit correspondre à une tension différentielle négative (Ux – Uy) qui à son tour doit correspondre à un niveau BAS du signal RxS d'un récepteur.

L'état de RxS n'est pas défini lorsque la ligne est en veille.

4.6.3.3 Sensibilité du récepteur

Un récepteur qui reçoit des trames comprenant 64 bits de données aléatoires, à une vitesse de 1 000 trames par seconde, ne doit pas détecter plus de trois erreurs de trame tous les $3 \times 3 \times 10^{+6}$ trames, lorsque l'amplitude du signal reçu définie en 4.6.3.1 varie entre ses valeurs minimale et maximale.

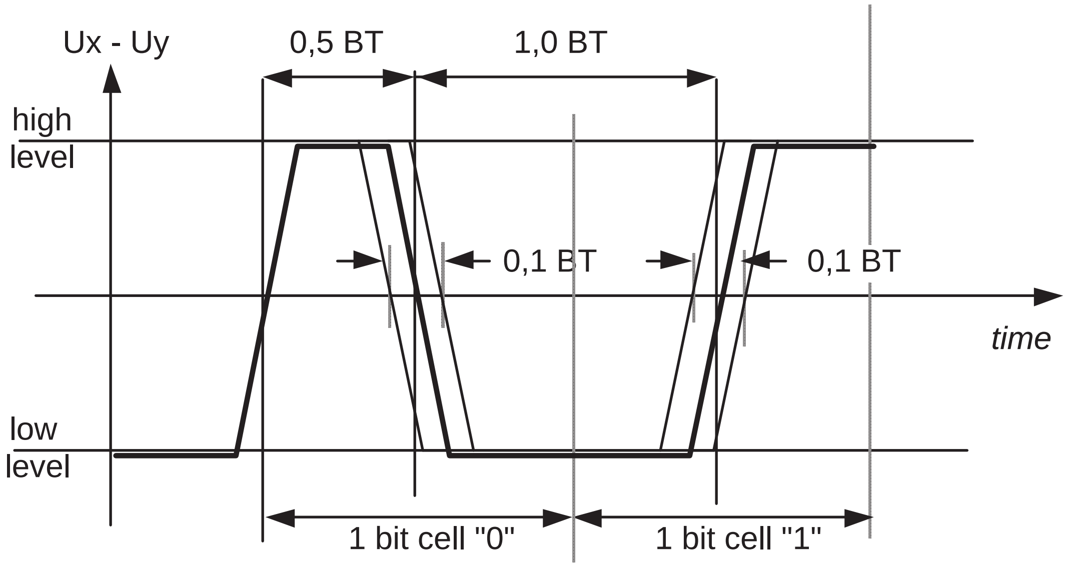
NOTE Selon la Figure 27, un récepteur peut fonctionner dans une plage de tensions comprise entre 5,00 V et 0,330 V, soit environ 23,6 dB. Cela laisse une marge de bruit d'environ 4,0 dB, étant donné que 4.2.3.3 spécifie que l'atténuation du support est inférieure à 20,0 dB.

4.6.3.4 Insensibilité du récepteur

Le récepteur ne doit pas décoder une trame valide (voir 4.7.1.5.3) lorsque le signal reçu (4.6.3.1) est inférieur à 0,100 V.

4.6.3.5 Distorsion frontale du récepteur

Un récepteur qui reçoit des trames comprenant 64 bits de données aléatoires, à une vitesse de 1 000 trames par seconde, ne doit pas détecter plus de trois erreurs de trame tous les $3 \times 3 \times 10^{+6}$ trames lorsque les flancs du signal d'essai passent par zéro de manière aléatoire 1,0 BT ± 10 % autour du passage prévu, comme illustré à la Figure 28.



Légende

Anglais	Français
high level	niveau élevé
low level	niveau bas
1 bit cell "0"	1 cellule de bit "0"
1 bit cell "1"	1 cellule de bit "1"
time	durée

Figure 28 – Distorsion frontale du récepteur

4.6.3.6 Rejet du bruit du récepteur

Un récepteur qui reçoit des trames comprenant 64 bits de données aléatoires, à une vitesse de 1 000 trames par seconde et une amplitude du signal de 0,700 V (1,4000 V crête à crête), ne doit pas détecter plus de 3 erreurs de trame tous les 3 $\times$ 10^{+6} trames en fonctionnement:

- lorsqu'un signal sinusoïdal en mode commun avec une amplitude de 4,000 Veff et à une fréquence comprise entre 65,0 Hz à 1,5 MHz est appliqué entre le boîtier et les deux fils de données; ou
- en présence d'un bruit gaussien quasi blanc additif (appliqué entre X et Y) réparti sur une bande passante de 1,0 kHz à 4,0 MHz, avec une amplitude de 0,140 Veff.

4.7 Signalisation dépendant du support

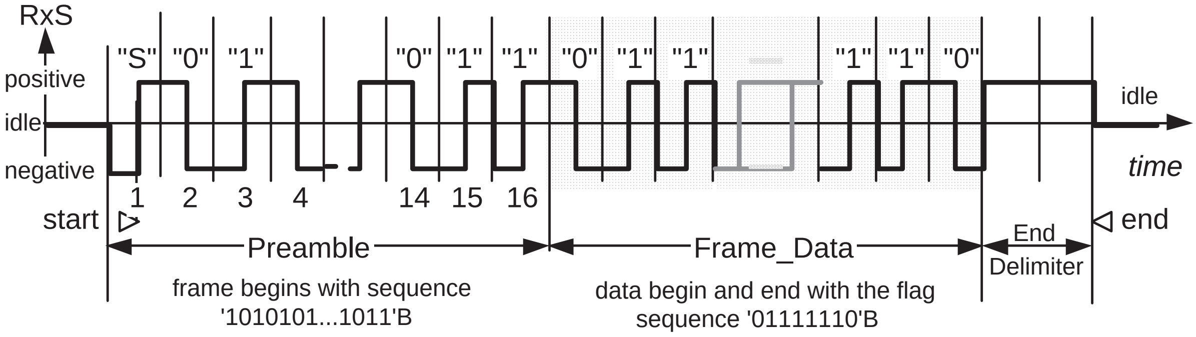
4.7.1 Codage et décodage des trames

4.7.1.1 Conventions

Le codage et le décodage se font en supposant que le signal émis ou le signal reçu est binaire, sans niveau polarisé. Le niveau de réception n'est pas défini lorsque la ligne est en veille.

RxS représente le signal idéal (analogique) reçu de la ligne.

Une trame doit être émise comme une séquence de niveaux positifs et négatifs commençant par un Préambule et se terminant par un Délimiteur de Fin, avant de retourner à l'état de veille (voir la Figure 29).



Légende

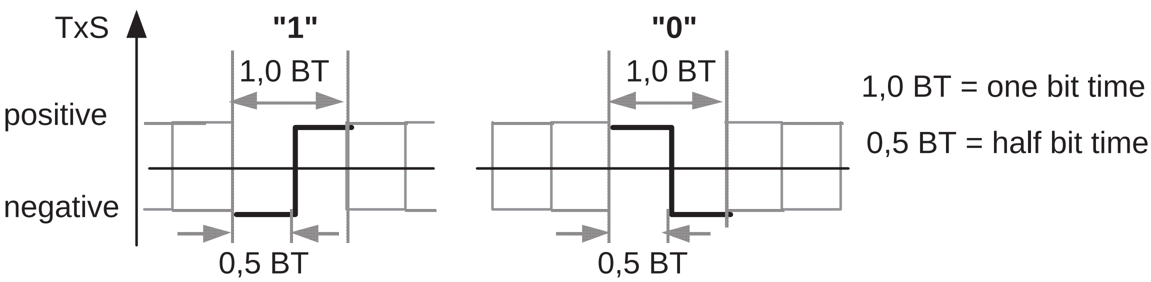
Anglais	Français
positive	positif
idle	veille
negative	négatif
Preamble	Préambule
time	durée
frame begins with sequence '1010101…1011'B	la trame commence par la séquence '1010101…1011'B
data begin and end with the flag sequence '01111110'B	les données commencent et se terminent par la séquence de drapeau '01111110'B
end delimiter	délimiteur de fin

Figure 29 – Trame idéale sur la ligne (avec un Préambule de 16 bits)

4.7.1.2 Codage des bits

Les bits Préambule et de trame Frame_Data doivent être codés selon le code Manchester (voir la Figure 30):

- un bit ‘1’ doit être codé par un niveau négatif pendant la première moitié d'une cellule de bit pour passer au niveau positif en milieu de cellule;
- un bit ‘0’ doit être codé par un niveau positif pendant la première moitié d'une cellule de bit pour passer au niveau négatif en milieu de cellule.



Légende

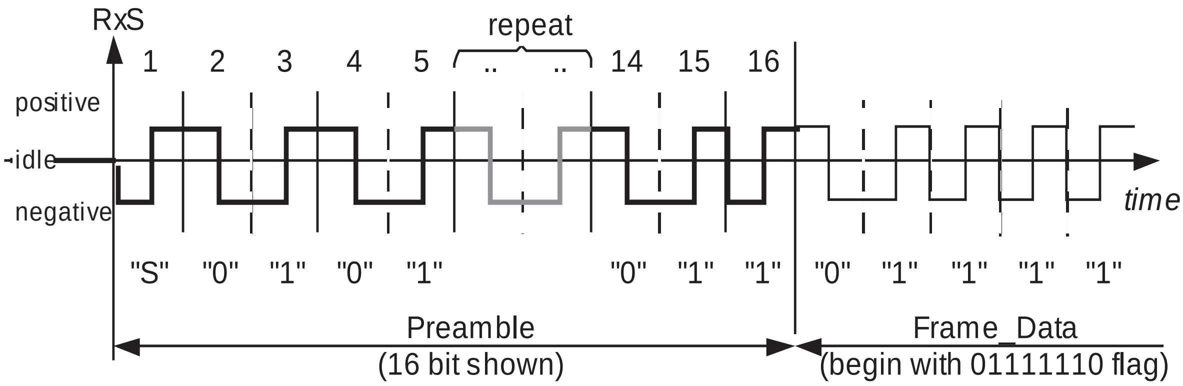
Anglais	Français
positive	positif
negative	négatif
one bit time	durée d’un bit
half bit time	durée d’un demi-bit

Figure 30 – Codage des bits

La distance entre les flancs doit être soit un temps de bit complet (‘0’ suit ‘1’ ou ‘1’ suit ‘0’), soit un demi-temps de bit (une séquence de ‘0’ ou de ‘1’) jusqu'au Délimiteur de Fin de la trame.

4.7.1.3 Codage du préambule

Une trame doit commencer par un Préambule comprenant une séquence de bits commençant par un bit de début S émis comme un bit ‘1’, suivi de paires de bits (‘0’ et ‘1’) et se terminant par un bit ‘1’ comme représenté à la Figure 31.



Légende

Anglais	Français
repeat	répétition
Preamble	Préambule
16 bit shown	16 bits
(begin with 01111110 flag)	(commence par le drapeau 01111110)
Time	Durée
positive	positif
idle	veille
negative	négatif

Figure 31 – Préambule

Le bit du début S et le bit de clôture ‘1’ doivent être séparés par au minimum 7 et au maximum 15 paires de bits (‘0’, ‘1’).

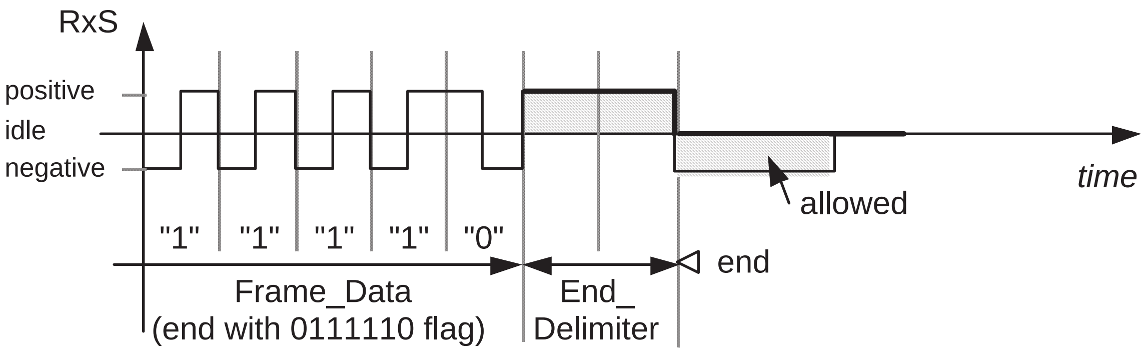
Le décodeur peut vérifier la polarité de RxS en décodant le Préambule, mais il ne doit pas inverser automatiquement le signal, si X et Y ont été inversés de manière intempestive.

NOTE Dans ce qui suit, seul un Préambule à 16 bits est considéré.

4.7.1.4 Délimiteur de Fin

La trame doit être fermée par un Délimiteur de Fin qui maintient le niveau positif de la ligne pendant 2,0 BT.

Un niveau négatif de 2,0 BT d'une durée de 2,0 BT peut être ajouté au niveau positif pour compenser le déséquilibre (voir la Figure 32).



Légende

Anglais	Français
allowed	admis
end	fin
time	durée
(end with 0111110 flag)	(se termine par le drapeau 0111110)
positive	positif
allowed	admis
idle	veille
negative	négatif

Figure 32 – Délimiteur de Fin

NOTE 1 Un Délimiteur de Fin sans niveau négatif provoque un déséquilibre qui conduit la ligne à résonner (voir 4.6.2.4). L'impulsion de compensation est vivement recommandée, mais elle n'est pas obligatoire, pour permettre l'utilisation de circuits disponibles sur le marché qui ne la génèrent pas.

NOTE 2 Le dernier bit d'une trame est un '0' à cause du drapeau HDLC (voir 5.2.1).

4.7.1.5 Contrôle de la qualité du signal

Les spécifications suivantes supposent que le décodeur génère deux signaux appelés Carrier_Sense (CS) et Signal_Quality_Error (SQE), pour contrôler la qualité du signal et la commutation de redondance.

4.7.1.5.1 Carrier_Sense

Le décodeur doit activer CS dans les 0,5 BT suivant la détection du dernier bit d'un Préambule reçu (voir 4.7.1.3).

Le décodeur doit désactiver CS dans les 0,5 BT suivant la détection d'un Délimiteur de Fin ou des bits qui ne sont ni '0', ni '1', ni un Délimiteur de Fin.

4.7.1.5.2 Signal_Quality_Error

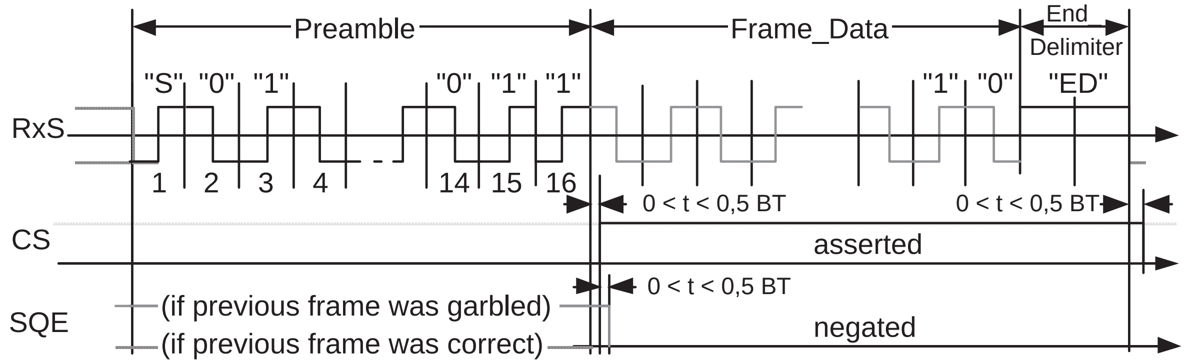
Le décodeur doit désactiver SQE (Signal_Quality_Error) dans les 0,5 BT après avoir détecté le dernier bit d'un Préambule reçu (voir 4.7.1.3).

Le décodeur doit activer SQE (Signal_Quality_Error) dans les 0,5 BT après avoir détecté des bits qui ne sont ni des '0', ni des '1', ni un Délimiteur de Fin pendant que CS est activé.

4.7.1.5.3 Trame valide

Une trame doit être considérée comme valide si elle comprend un Préambule, un certain nombre de bits '0' et '1', ainsi qu'un Délimiteur de Fin.

EXEMPLE La Figure 33 illustre une trame valide avec les signaux CS et SQE correspondants.



Légende

Anglais	Français
Preamble	Préambule
asserted	activé
negated	désactivé
End Delimiter	Délimiteur de Fin
(if previous frame was garbled)	(si la trame précédente était brouillée)
(if previous frame was correct)	(si la trame précédente était correcte)

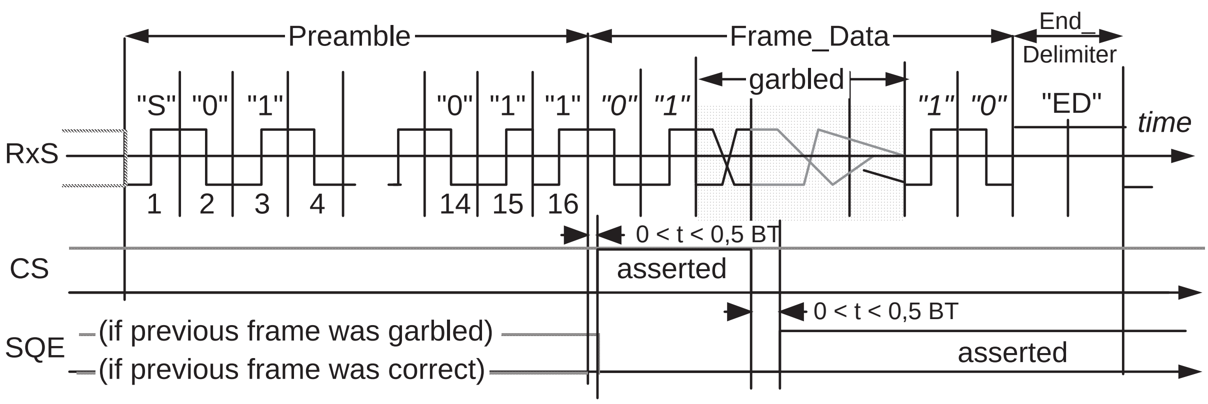
Figure 33 – Trame valide avec les signaux RxS, CS et SQE

Pour le contrôle de la redondance, une trame valide doit comprendre un Préambule suivi d'une séquence d'au moins huit bits de données.

4.7.1.5.4 Trame non valide

Une trame doit être considérée comme non valide si SQE est activé pendant plus de 0,5 BT pendant que CS est actif.

EXEMPLE La Figure 34 illustre les signaux lors de la réception d'une trame brouillée.



Légende

Anglais	Français
Preamble	Préambule
garbled	brouillé
time	durée
asserted	activé
(if previous frame was garbled)	(si la trame précédente était brouillée)
(if previous frame was correct)	(si la trame précédente était correcte)
End Delimiter	Délimiteur de Fin

Figure 34 – Trame brouillée avec les signaux RxS, CS et SQE

Si SQE devient actif, le décodeur doit ignorer toutes les données jusqu'à la réception du Préambule suivant.

4.7.2 Traitement de lignes doublées (en option)

La présente spécification définit un schéma de redondance facultatif. Lorsque cette option est utilisée, les spécifications suivantes s'appliquent à la Direction_1 et la Direction_2 ainsi qu'à la Line_A et la Line_B.

4.7.2.1 Principe

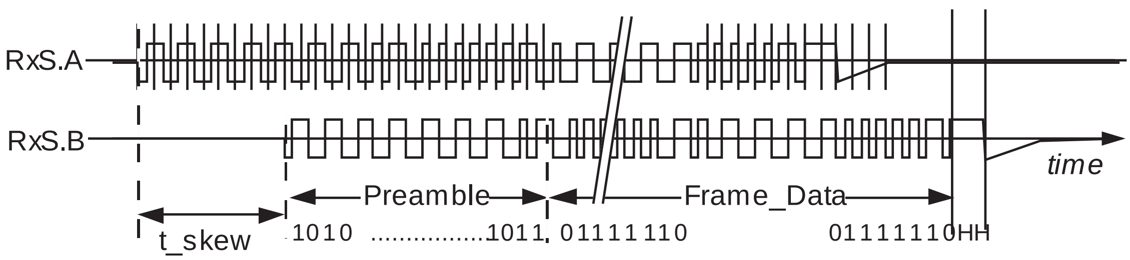
Un nœud émet les mêmes données en même temps sur la Line_A et la Line_B. Un nœud accepte les données d'une ligne appelée Trusted_Line, tout en contrôlant l'autre ligne appelée Observed_Line.

Chaque nœud sélectionne ses lignes Trusted_Line et Observed_Line, indépendamment des autres nœuds, en fonction des signaux générés par sa propre couche physique ou à la demande de sa couche de liaison.

Pour rester indépendant du support, la sélection de Trusted_Line dépend des signaux générés par l'Unité de Ligne, selon la définition dans l'interface Unité de Ligne.

4.7.2.2 Décalage

Etant donné que les signaux de Line_A et de Line_B font l'objet de délais différents, leur décalage (différence de temps) diffère au niveau de l'émetteur, du récepteur ou n'importe où sur la ligne (voir la Figure 35).



Légende

Anglais	Français
Preamble	Préambule
time	durée

Figure 35 – Lignes redondantes (vues par un récepteur)

4.7.2.3 Transmission redondante

Une MAU avec des Unités de Ligne redondantes doit émettre le même signal sur Line_A et Line_B (Line_A1 et Line_B1 ou Line_A2 et Line_B2).

La différence de temps entre le signal mesuré à la sortie des Unités de Ligne entre Line_A et Line_B dans la même direction ne doit pas dépasser T_skew_t = 1,0 μs.

4.7.2.4 Réception redondante

4.7.2.4.1 Décalage à la réception

Un récepteur doit admettre un décalage maximal de T_skew_r = 32,0 μs.

4.7.2.4.2 Line_Disturbance (Perturbation de Ligne)

Il doit y avoir un signal Line_Disturbance pour chaque section du bus raccordée à un noeud, appelés DA1 et DA2 pour Line_A et dB1 et dB2 pour Line_B.

Le signal Line_Disturbance d'une ligne doit être activé si:

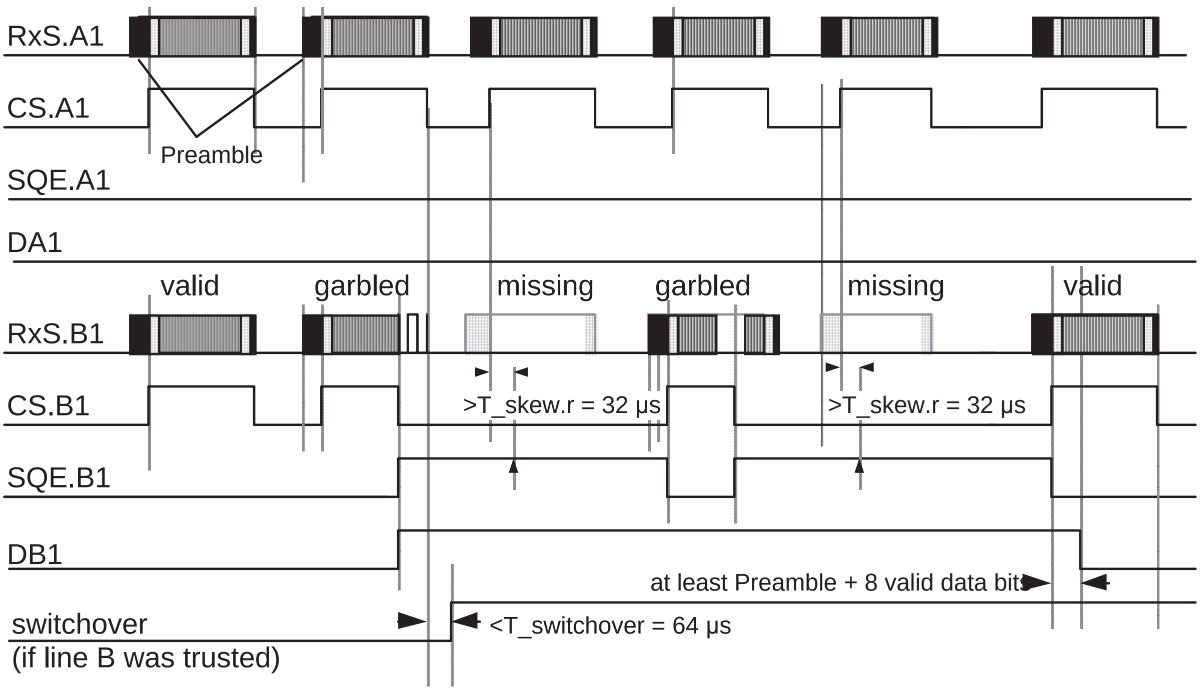
- le décodeur active ‘Signal_Quality_Error’ sur cette ligne; ou
- si le décodeur ne génère pas un ‘Carrier_Sense’ dans T_skew_r après que l’Unité de Ligne de la ligne redondante ait activé ‘Carrier_Sense’ (trame manquante).

Le signal Line_Disturbance doit être désactivé:

- lorsque le décodeur reçoit une trame valide (voir 4.7.1.5.3).

En mode non redondant, la ligne non utilisée doit être considérée comme perturbée en permanence.

EXEMPLE La Figure 36 est une illustration de la validité de toutes les trames sur la Line_A1 tandis que la Line_B1 est perturbée.



Légende

Anglais	Français
Preamble	Préambule
valid	valide
garbled	brouillé
at least Preamble + 8 valid data bits	au moins Préambule + 8 bits de données valides
switchover	commutation
(if line B was trusted)	(si la ligne B a été considérée fiable)
missing	manquant

Figure 36 – Signaux de Line_Disturbance

4.7.2.5 Commutation

Direction_1 et Direction_2 d'un nœud (d'extrémité) peuvent se fier à des lignes différentes (Line_A1 et Line_B2, par exemple).

L'unité de commutation doit échanger Trusted_Line et Observed_Line dans un intervalle de T_switchover = 64,0 μs:

a) si le signal Line_Disturbance de Trusted_Line est activé pendant que le signal Line_Disturbance de Observed_Line reste désactivé;
b) à la demande de la Couche de Liaison (spécialement en cas d'une erreur de taille, de FCS ou de protocole).

4.7.2.6 Rapport de la MAU

La MAU doit rendre compte au gestionnaire de liaison:

a) de la ligne sur laquelle elle a reçu une trame;
b) de chaque activation du signal Line_Disturbance sur chaque ligne et dans chaque direction;
c) de l'état des quatre signaux Line_Disturbance (DA1, DA2, dB1, dB2) pour Node_Report (voir 5.5.2.2).

NOTE Si une ligne est complètement hors service (ou si elle n'est pas reliée), le signal Line_Disturbance est activé une seule fois après l'apparition du défaut. Cependant, si la ligne est interrompue, le signal Line_Disturbance peut être à nouveau activé, puis sésactivé après presque chaque trame, selon la localisation de l'interruption.

4.7.3 Interface de l'Unité de Ligne

L'interface Unité de Ligne définit les signaux d'entrée et de sortie d'une Unité de Ligne.

Cette interface peut rester à l'intérieur d'un nœud, mais il convient de la rendre accessible pour les essais. Cette interface ne fait pas l'objet d'essais de conformité.

La spécification suivante facilite l'interfaçage et définit les signaux utilisés dans les autres paragraphes de la présente norme.

Si elle est accessible, l'Interface Unité de Ligne doit comprendre des signaux de modem selon la définition de la Recommandation V.24 de l'UIT-T, pour chaque émetteur-récepteur individuellement, et de signaux de contrôle supplémentaires (voir le Tableau 7).

Tableau 7 – Signaux de l'Interface Unité de Ligne

Nom	Désignation	Article de la Rec. V.24 de l'UIT-T	Direction	Signification
GND	Masse des signaux	102	-	retour commun
TxD	Données de l'émetteur	103	vers l'Unité de Ligne	Frame_Data, sans Préambule ni Délimiteur de Fin, émis comme un signal NRZ, synchronisé par TxC
RxD	Données du récepteur	104	en provenance de l'Unité de Ligne	séquence NRZ, sans Préambule ni Délimiteur de Fin, synchronisé par RxC
RTS	Demande d'émission	105	vers l'Unité de Ligne	commandes d'envoi du Préambule, la désactivation de ce signal commande la génération du Délimiteur de Fin.
CTS	Prêt à émettre	106	en provenance de l'Unité de Ligne	signale que le Préambule a été émis et demande les données à suivre
TxC	Horloge d'émission	114	en provenance de l'Unité de Ligne	généré par l'émetteur-récepteur pour synchroniser les données TxD à émettre
RxC	Horloge de réception	115	en provenance de l'Unité de Ligne	généré par le décodeur pour synchroniser les données RxD reçues.
SQE	Signal_Quality_Error	Non V.24	en provenance de l'Unité de Ligne	selon les spécifications de 4.7.1.5.2
CS	Carrier_Sense	Non V.24	en provenance de l'Unité de Ligne	selon les spécifications de 4.7.1.5.1
Kx	signaux de commande de commutation	Non V.24	vers l'Unité de Ligne	définissent au moins les deux positions de commutation de base: End_Setting et Intermediate_Setting, pour les deux lignes. De plus, les signaux de commande de commutation peuvent isoler un émetteur-récepteur de la ligne ou le relier à la ligne.

5 Contrôle de la Couche de Liaison

5.1 Adressage

La Couche de Liaison doit utiliser un indicatif de 8 bits pour la Device_Address de source et de destination.

Le Canal Principal d'un nœud doit être adressé par une adresse de dispositif dans la plage 1 ('00000001'B) à 63 ('00111111'B) attribuée par la procédure d'inauguration.

Le Canal Principal du maître doit recevoir l'adresse 'maître' 1 ('00000001'B).

L'adresse de dispositif 0 ('00000000'B) doit être l'adresse propre du dispositif et ne doit pas être émise.

Les adresses de dispositif 64 à 126 et 128 à 254 sont réservées pour une utilisation ultérieure.

L'adresse 255 ('1111 1111'B) doit être celle de la 'diffusion', que tous les nœuds écoutent.

Un nœud non nommé doit répondre à l'adresse 127 ('01111111'B) sur ses deux canaux.

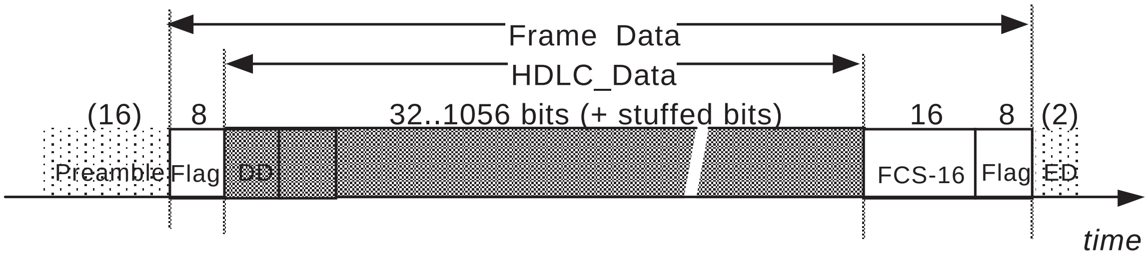
Le Canal Auxiliaire d'un nœud doit répondre à l'adresse ‘sans nom’.

Une adresse ‘nœud’ est une adresse esclave ou maître nommée.

5.2 Trames et télégrammes

5.2.1 Format des Données de Trame (Frame_Data)

Le format Frame_Data doit être conforme au format HDLC défini par l'ISO/CEI 13239 (voir la Figure **37).**



Légende

Anglais	Français
(+ stuffed bits)	(+ bits de bourrage)
Preamble	Préambule
Flag	Drapeau
time	durée

Figure 37 – Structure de la trame HDLC

Une trame doit commencer par un seul drapeau composé d'un bit ‘0’ suivi de six bits ‘1’ contigus et fermé par un bit ‘0’ (voir l'ISO/CEI 13239).

Le drapeau d’ouverture doit être suivi des Données HDLC comprenant au moins 32 bits et au plus1056 bits (sans tenir compte des bits de bourrage prévus par l'ISO/CEI 13239).

Les Données HDLC doivent comprendre un nombre entier d'octets (en restriction par rapport à l'ISO/CEI 13239).

Le premier octet des Données HDLC doit être le Dispositif Destinataire à 8 bits spécifié en 5.1.

Le deuxième octet des Données HDLC ne doit pas être interprété comme le champ de contrôle de l'ISO/CEI 13239.

Les Données HDLC doivent être suivies d'un code de détection d'erreurs qui doit être la Séquence de Contrôle de Trame (FCS) de 16 bits dont la structure est conforme à l'ISO/CEI 13239.

La trame doit être fermée par un seul drapeau de fermeture identique au drapeau d'ouverture.

Un drapeau de fermeture ne peut être utilisé comme le drapeau d'ouverture de la trame suivante.

L'émetteur ne doit pas émettre les séquences ‘idle’ ni ‘cancel’ de l'ISO/CEI 13239.

Une trame est considérée comme une séquence d'octets. Le bit de poids le plus faible de chaque octet doit être émis en premier, comme le prévoit l'ISO/CEI 13239.

NOTE 1 L'ordre de transmission vaut uniquement pour la séquence de bits à l'intérieur d'un octet. Les deux octets FCS sont émis avec le bit de poids le plus fort en premier à cause d'une exception prévue par l'ISO/CEI 13239.

NOTE 2 Le bourrage de bits des données HDLC empêche l'apparition de séquences drapeaux dans les données entre les deux drapeaux: l'émetteur insère un '0' après chaque groupe de cinq '1' consécutifs dans les données. Le récepteur enlève le '0' qui suit un groupe de cinq '1'. Les bits de bourrage sont invisibles à la couche de liaison. Ils peuvent cependant augmenter la taille d'une trame de 20 %.

5.2.2 Cadence des télégrammes

5.2.2.1 Conventions

Un segment de bus doit être commandé par un nœud (le maître) qui peut émettre sur le bus à son gré. Les autres nœuds sont des nœuds Esclaves qui ne peuvent émettre qu'à la demande du maître.

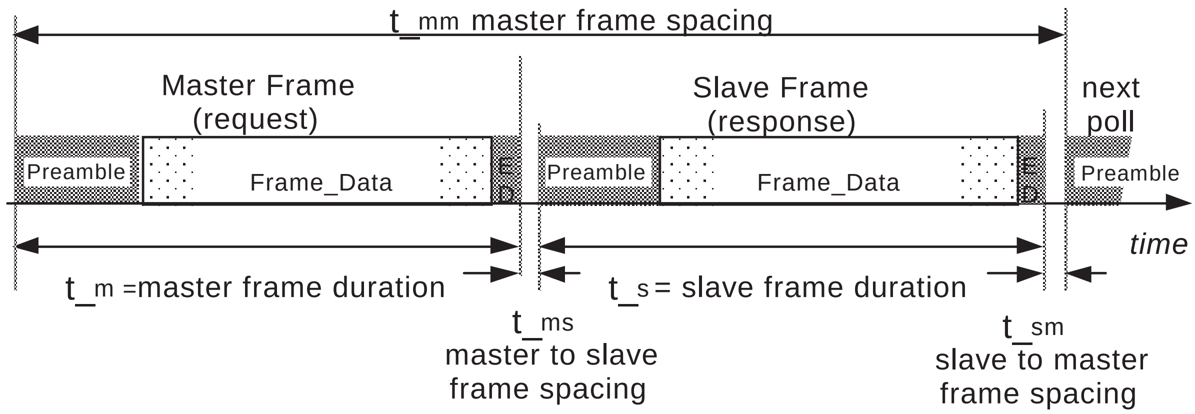
Un Nœud d'Extrémité est considéré comme le maître du Canal Auxiliaire.

A l'intérieur d'un nœud, les fonctions maître et esclave sont distinctes.

Le flux de données sur le bus doit comprendre des paires de trames (les télégrammes) qui comprennent une Trame-Maître envoyée par le maître, et auquel un esclave répond en envoyant une Trame-Esclave dans un temps donné.

L'intervalle entre les trames doit être mesuré à partir de la dernière transition d'un Délimiteur de Fin jusqu'à la transition centrale du bit de départ du Préambule.

EXEMPLE La Figure 38 illustre la cadence d'un télégramme.



Légende

Anglais	Français
t_mm master frame spacing	Intervalles entre les trames du maître t_mm
Master Frame (request)	Trame Maître (demande)
Slave Frame (response)	Trame Esclave (réponse)
next poll	interrogation suivante
Preamble	Préambule
time	durée
master frame duration	durée de la trame maître
slave frame duration	durée de la trame esclave
slave to master frame spacing	intervalle entre les trames esclave et maître
master to slave frame spacing	intervalle entre les trames maître et esclave

Figure 38 – Cadence d'un télégramme

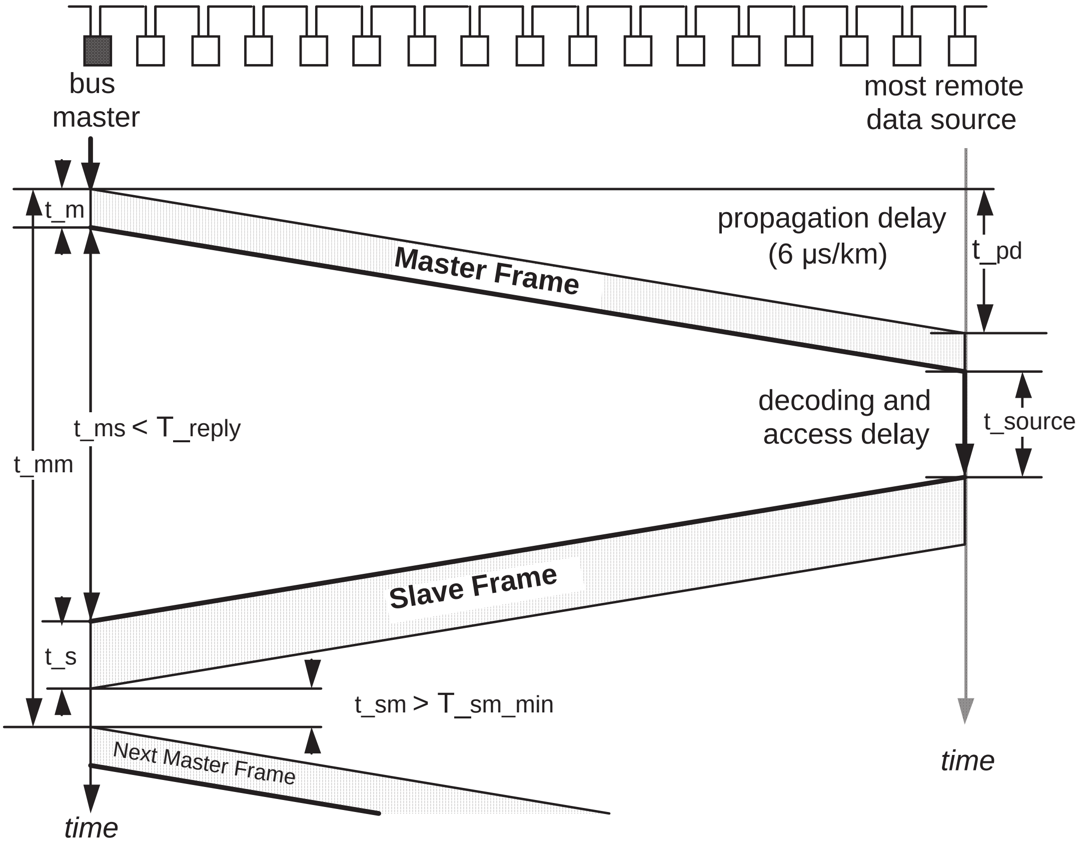
5.2.2.2 Calcul du temps de réponse

Pour un bus donné, le temps de réponse T_reply est le délai maximal entre la fin d'une Trame-Maître et le début de la Trame-Esclave envoyée en réponse, mesurée au niveau du maître.

Le temps de réponse est la somme des temps de propagation, de décodage et d'accès.

T_reply est un paramètre de configuration qui signale au maître le temps d'attente avant d'émettre la Trame-Maître suivante s'il n'a pas reçu de Trame-Esclave.

EXEMPLE La Figure 39 illustre une configuration de 17 nœuds (16 sections) en supposant que le maître se trouve à une extrémité du bus.



Légende

Anglais	Français
propagation delay	délai de propagation
Master Frame	Trame Maître
decoding and access delay	délai de décodage et d'accès
Slave Frame	Trame Esclave
Next Master Frame	Trame Maître suivante
time	durée
bus master	maître de bus
most remote data source	source de données la plus éloignée

Figure 39 – Exemple d'intervalle entre les trames

Le temps de réponse T_reply le plus défavorable pour une application donnée est défini comme suit:

T_reply [μs] = 2 $\times$ T_pd + T_source_max

- T_source_max vaut pour le décodage de la Trame-Maître et la réponse à la source (voir 5.2.2.4.2);
- T_pd est le temps de propagation le plus défavorable pour une trame entre les Nœuds d'Extrémité d'une application donnée (voir 4.2.3).

Les temporisateurs d'une trame sont calculés comme suit:

Temps de bits	Canal principal	Canal auxiliaire
données d'application	1 024 bits	16 bits
en-tête	32 bits	32 bits
CRC	16 bits	16 bits
nombre total de bits	1 072 bits	64 bits
bourrage de bits (au pire $\times$ 1,2)	1 286 bits	77 bits
2 $\times$ drapeau	16 bits	16 bits
délimiteur de fin	2 bits	2 bits
taille maximale du préambule	32 bits	32 bits
nombre total de bits	1 336 bits	127 bits
durée à 1,0 Mbit/s	1 336,0 μs	127,0 μs
2 $\times$ délai de propagation	120,0 μs	120,0 μs
temps de réponse de l'esclave	300,0 μs	800,0 μs
Total	1 756,0μs	1 047,0μs

5.2.2.3 Collision

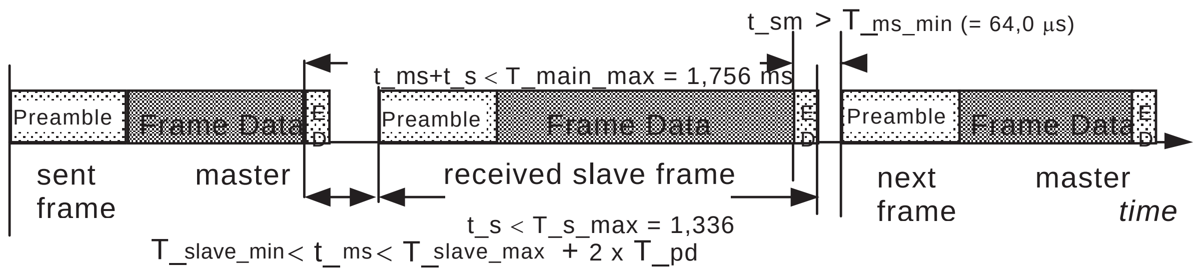
Une collision se produit quand plusieurs émetteurs sont actifs simultanément. Cette situation n'apparaît normalement qu'entre les Nœuds d'Extrémité de segments de bus différents qui émettent en même temps. Il n'est fait aucune distinction entre les collisions et les trames perdues ou non valides.

5.2.2.4 Intervalles entre les trames émises

5.2.2.4.1 Côté maître

Sur le Canal Principal, un maître doit s'attendre à recevoir entièrement une Trame-Esclave émise en réponse à sa Trame-Maître pendant un délai T_main_max = 1,756 ms depuis la fin de sa Trame-Maître émise. Il peut envoyer sa Trame-Maître suivante T_sm_min = 0,064 ms après la fin de la Trame-Esclave reçue ou à l'échéance du temporisateur (1,756 ms + 0,064 ms = 1,820 ms) (voir la Figure 40).

Sur le Canal Auxiliaire, un Nœud d'Extrémité doit s'attendre à recevoir entièrement une Trame-Esclave en réponse à sa Trame-Maître pendant un délai T_aux_max = 1,047 m. Il peut envoyer sa Trame-Maître suivante T_ms_min = 0,064 ms après la fin de la Réponse de Détection (Detect_Response) reçue ou à l'échéance du temporisateur (1,047 ms + 0,064 ms = 1,111 m**s).**



Légende

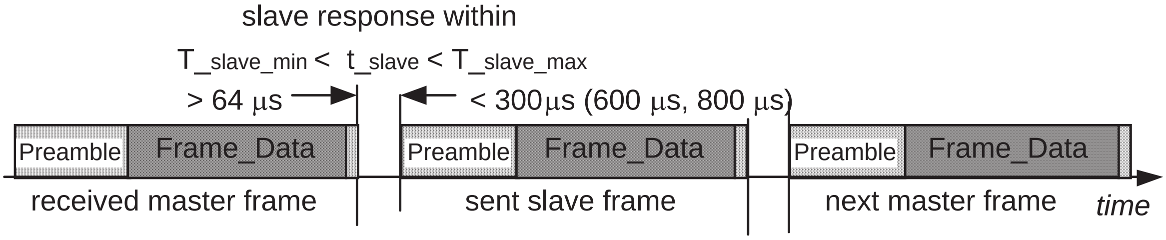
Anglais	Français
Preamble	Préambule
Frame Data	Données de Trame
sent master frame	trame maître envoyée
received slave frame	trame esclave reçue
next master frame	trame maître suivante
time	durée

Figure 40 – Intervalles entre les trames mesurés du côté du maître

5.2.2.4.2 Côté esclave

L'esclave adressé doit commencer à émettre sa trame (voir la Figure 41):

- pas plus tôt que T_source_min = 64,0 µs après avoir reçu la fin de la Trame-Maître; et
- pas plus tard que T_source_max = 0,300 ms après avoir reçu la fin de la Trame-Maître, sauf dans le cas d'une Réponse de Détection (Detect_Response) pour laquelle ce temps est augmenté à 0,600 ms pendant une inauguration et à 0,800 ms en fonctionnement normal.



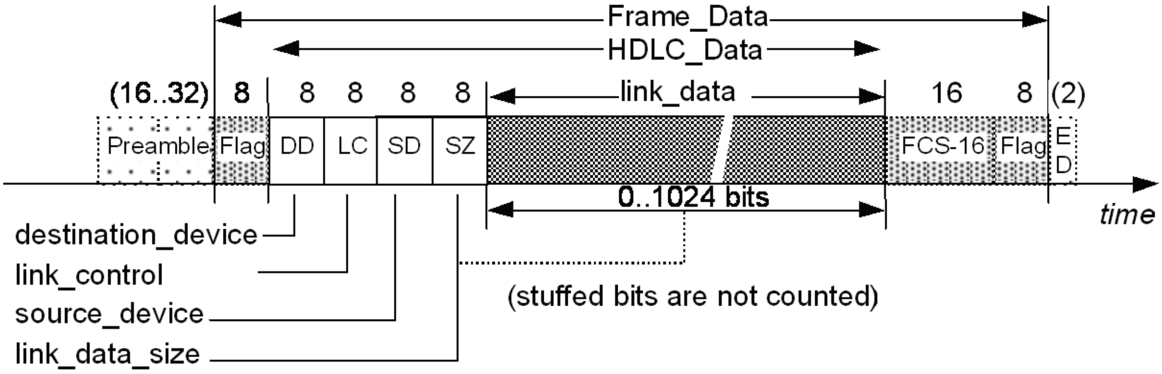
Légende

Anglais	Français
slave response within	réponse de l'esclave dans les
sent master frame	trame maître envoyée
received slave frame	trame esclave reçue
next master frame	trame maître suivante
time	durée
Preamble	Préambule

Figure 41 – Intervalles entre les trames mesurés du côté de l'esclave

5.2.3 Élément d'une trame HDLC

Les Données HDLC sont composées de champs compris entre le Drapeau d'ouverture et la séquence de contrôle, en excluant les bits de bourrage (voir la Figure 42).



Légende

Anglais	Français
time	durée
(stuffed bits are not counted)	(les bits de bourrage ne sont pas comptés)
Preamble	Préambule
Flag	Drapeau

Figure 42 – Format des Données HDLC

Les Données HDLC doivent se composer des champs suivants (voir la Figure 43).

```
HDLC_Data::= RECORD
  {
  destination_device    UNSIGNED8          -- adresse ‘noeud’ à 8 bits du
                                              récepteur ou adresse
                                              ’diffusion’; dans une trame
                                              de réponse, cette adresse
                                              est par défaut celle du
                                              maître.
  link_control          Link_Control       -- Link_Control à 8 bits
  source_device         UNSIGNED8          -- adresse ‘noeud’ à 8 bits de
                                              l’émetteur de la trame;
                                              dans une trame de requête,
                                              cette adresse est par défaut
                                              l’adresse ‘maître’.
  link_data_size        UNSIGNED8          -- taille à 8 bits des
                                              Données de Liaison qui
                                              suivent ce champ, exprimée
                                              par un nombre entier
                                              d’octets; cette taille doit
                                              être ‘0’ quand le champ
                                              ‘Link_Data’ est vide;
  link_data             Link_Data          -- ARRAY [link_data_size] OF
                                              WORD8
                                              entre 0 et 1024 bits de
                                              données; Voir 5.3.1 pour
                                              plus de détails.
}
```

0	1	2	3	4	5	6	7
destination_device (DD)							
link_control (LC)							
source_device (SD)							
link_data_size (SZ)							
link_data							
entre 0 et 1024 bits							

NOTE Les alias DD, LC, SD, SZ sont employés dans les figures pour gagner de la place.

Figure 43 – Format de données HDLC

5.2.4 Champ de Contrôle de Liaison (Link_Control)

Le champ de Link_Control distingue:

a) les Requêtes (Trames-Maîtres);

b) les Réponses (Trames-Esclaves).

Le champ de Link_Control distingue trois types de télégrammes:

c) les télégrammes de Données de Processus permettent de mettre à jour la base de données de Processus;

d) les télégrammes de Données de Messagerie servent au transfert de messages;

e) les télégrammes de Données de Supervision servent à la supervision et à l’inauguration du bus.

Le champ de Link_Control permet aux nœuds scrutés de signaler une demande d’émission spontanée, un changement d’état ou des conditions d’inauguration grâce à quatre bits:

f) ‘A_bit’: (Attention) émission de Données de Messagerie demandée;

g) ‘C_bit’: (Change) changement d'état du nœud;

h) ‘I_bit’: (Inhibit) inauguration non autorisée;

i) ‘RI_bit’: (Remote Inhibit) inauguration à distance non autorisée.

Le champ de Link_Control doit être codé sur 8 bits (voir le Tableau 8).

Tableau 8 – Codage de Link_Control

		Codage							
	Type de Trame	7	6	5	4	3	2	1	0
Données de Processus et	Process_Data_Request / Process_Data_Response	M	0	A	C	I	0	0	0
Données de Messagerie	Message_Data_Request/Message_Data_Response	M	0	A	C	0	1	1	1
Données de Supervision	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

Les combinaisons de bits qui ne sont pas spécifiées dans le Tableau 8 sont réservées et doivent être ignorées.

Le champ de Link_Control est spécifié comme suit:

```
Link_Control::= RECORD
  {
  mq                       ENUM1                  -- bit de poids le plus fort
    {
    SR        (0),                                -- ‘0’ dans une réponse d’esclave
    MQ        (1)                                 -- ‘1’ dans une demande de maître
    }
  sup                      ENUM1
    {
    PM        (0),                                -- données de processus ou message
    SP        (1)                                 -- données de supervision
    },
ONE_OF [sup]
    [PM]                   RECORD
      {
      a_bit                BOOLEAN1,              -- ‘1’ si le A_bit est activé dans
                                                     Process_Data_Response ou
                                                     Message_Data_Response
      c_bit                BOOLEAN1,              -- ‘1’ si C_bit est activé dans
                                                     Process_Data_Response ou
                                                     Message_Data_Response
      i_bit                BOOLEAN1,              -- ‘1’ si I_bit est activé dans
                                                     Process_Data_Request ou
                                                     Process_Data_Response
      pom                  ENUM3
       {
       PROCESS_DATA        (0),                   -- Process_Data_Request ou
                                                     Process_Data_Response
       MESSAGE_DATA        (7)                    -- Message_Data_Request ou
                                                     Message_Data_Response
       }
      },
    [SP]                   RECORD                 -- Supervisory_Data_Request ou
                                                     Supervisory_Data_Response
      {
      res0                 WORD1 (0),             -- réservé, = 0
      rem_inh              BOOLEAN1               -- ‘1’ si ‘RI_bit’ est activé dans
                                                     Status_Response ou
                                                     Presence_Response
      i_bit                BOOLEAN1,              -- ‘1’ si ‘I_bit’ est activé dans
                                                     Presence_Request ou
                                                     Presence_Response
      supervisory_type     ENUM3                  -- distingue les
                                                     données de supervision
        {
        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 ou Topography_Response
        PRESENCE    (7)    -- Presence_Request ou Presence_Response
        }
      }
    }
  }
```

5.2.5 Traitement des bits 'Attention', 'Change' et 'Inhibit'

Process_Data_Responses et Message_Data_Responses doivent attribuer la valeur 1 aux bits suivants pour signaler les événements asynchrones:

a) A_bit (Attention) doit être activé aussi longtemps que la Queue d'Émission des Données de Messagerie contient des trames à émettre;
b) C_bit (Change) doit être activé pour signaler un changement du Node_Status. Il doit être désactivé lorsque le nœud reçoit un Status_Request;
c) I_bit (Inhibit) doit être défini de la manière suivante pour bloquer l'inauguration:
   - aussi longtemps qu'un nœud bloque l'inauguration, le noeud doit activer I_bit dans toutes ses trames de Données de Processus et de Données de Supervision (mais pas dans les Message_Data_Responses);
   - le maître doit copier la combinaison OU du I_bit présent dans Process_Data_Response reçu de chaque nœud qu'il a nommé, dans le I_bit de toutes les Presence_Requests qu'il envoie;
   - un Nœud d'Extrémité doit copier le I_bit reçu d'une Presence_Request dans son I_bit de ses Detect_Requests et Detect_Responses et dans le I_bit de ses Presence_Responses;
d) RI_bit (Remote_Inhibit) doit être activé de la manière suivante pour bloquer l'inauguration:
   - un Nœud d'Extrémité doit insérer le I_bit qu'il lit dans la Detect_Response d'une composition éloignée, dans le RI_bit de ses Presence_Responses et Status_Responses.

5.2.6 Erreurs de taille, de FCS et de protocole

Ce qui suit s'applique uniquement aux trames reçues avec l'adresse propre ou l'adresse de diffusion:

un récepteur doit ignorer une trame et considérer la ligne sur laquelle ladite trame a été reçue comme étant perturbée si:

a) sa Séquence de Contrôle de Trame (FCS) est erronée;
b) sa longueur ne correspond pas à celle déclarée dans le champ link_data_size;
c) s'il s'agit d'une Trame-Esclave qui n'est pas du même type (Données de Processus, Données de Messagerie, Données de Supervision) que la Trame-Maître qui la précède.

Le Canal Auxiliaire doit ignorer une trame et signaler une erreur de protocole si cette trame n'est pas du type: Detect_Request, Detect_Response ou Naming_Request.

Le Canal Principal doit ignorer une trame s'il s'agit d'une Detect_Request ou d'une Detect_Response.

Le maître doit ignorer une deuxième Trame-Esclave en réponse à une Trame-Maître (collision sans perturbation).

5.3 Formats et protocoles des télégrammes

5.3.1 Champ Link_Data

Le contrôle de la Couche de Liaison fait la distinction entre les Données de Processus, Données de Messagerie et les Données de Supervision.

Pour tenir compte des contraintes d'adressage, la définition des données HDLC comprend l'En-tête de Liaison:

```
HDLC_Data::= RECORD
    {
    ONE_OF [link_control.sup]              -- dépend du champ link_control
      {
      [PM] ONE_OF [link_control.pom]       -- Données de Processus ou
                                              Données de Messagerie
             {
             [PROCESS_DATA]
                ONE_OF [link_control.mq]   -- Demande ou Réponse
                  {
                  [MQ] Process_Data_Request,
                  [SR] Process_Data_Response
                  }
             [MESSAGE_DATA]
                ONE_OF [link_control.mq]   -- Demande ou Réponse
                  {
                  [MQ] Message_Data_Request,
                  [SR] Message_Data_Response
                  }
             },
      [SP] Supervisory Data
      }
    }
```

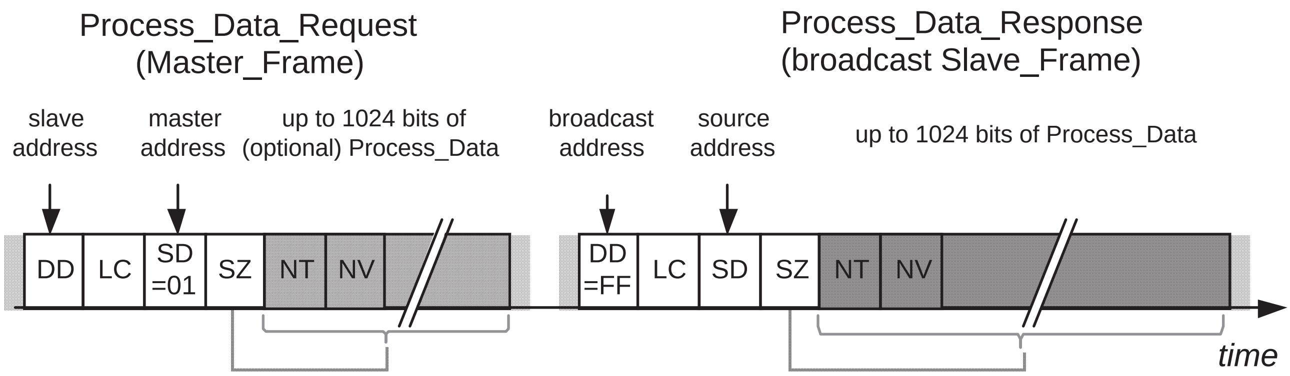
Les quatre premiers octets de Link_Data forment le Link_Header (En-tête de Liaison), leurs format et signification sont identiques dans toutes les trames.

5.3.2 Données de Processus

5.3.2.1 Action

Un maître doit demander la transmission des Données de Processus d'un autre nœud (ou de lui-même) ou envoyer des Données de Processus vers un nœud (en option) en utilisant une Process_Data_Request, auquel le nœud adressé doit répondre avec une trame Process_Data_Response, diffusée à tous autres les nœuds.

Un télégramme de Données de Processus se compose d'une Process_Data_Request suivie d'une Process_Data_Response (voir la Figure 44).



Légende

Anglais	Français
(broadcast Slave_Frame)	(Trame_Esclave de diffusion)
slave address	adresse de l'esclave
master address	adresse du maître
up to 1024 bits of (optional) Process_Data	Process_Data de 1024 bits au maximum (facultatif)
broadcast address	adresse de diffusion
source address	adresse source
up to 1024 bits of Process_Data	Process_Data de 1024 bits au maximum
time	durée

Figure 44 – Télégramme de Données de Processus

5.3.2.2 Process_Data_Request

Le format de Process_Data_Request doit se présenter comme suit (voir la Figure 45):

```
Process_Data_Request::= RECORD
  {
  destination_device    UNSIGNED8           -- adresse ‘noeud’ ou adresse
                                               ‘maître’ pour une demande du
                                               maître faite à lui-même
  link_control          Link_Control        -- Process_Data_Request
  source_device         UNSIGNED8           -- adresse ‘maître’
  link_data_size        UNSIGNED8           -- = 0 ou (en option)
                                               = (0 < link_data_size ≤128)
  ARRAY [link_data_size] OF WORD8           -- Process Data (option)
  }                                         -- contenu défini par
                                               l’application
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							
ARRAY [link_data_size] OF							
WORD8							

Figure 45 – Format de Process_Data_Request

5.3.2.3 Process_Data_Response

Le format de Process_Data_Response doit se présenter comme suit (voir la Figure 46):

```
Process_Data_Response::= RECORD
  {
  destination_device    UNSIGNED8           -- destination_device =
                                               diffusion
  link_control          Link_Control        -- Process_Data_Response
  source_device         UNSIGNED8           -- adresse ‘noeud’ ou ‘maître’
  link_data_size        UNSIGNED8           -- (0 ≤ link_data_size ≤128)
  ARRAY [link_data_size] OF WORD8           -- contenu défini par
                                               l’application;
                                               il est recommandé que les
                                               deux premiers octets soient
                                               le ‘Node_Key’ et que
                                               l’application vérifie que
                                               celui-ci correspond au
                                               Node_Key du noeud source tel
                                               qu’il a été reçu avec la
                                               Topographie.
  }
```

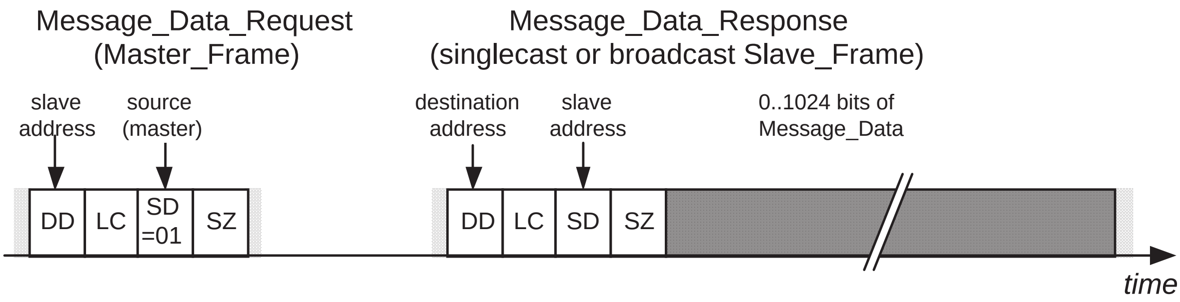
7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							
ARRAY [link_data_size] OF							
WORD8							

Figure 46 – Format de Process_Data_Response

5.3.3 Données de Messagerie

5.3.3.1 Action

Un maître doit demander à un autre nœud (ou à lui-même) de transmettre des Données de Messagerie à l’aide de Message_Data_Request, auquel le nœud adressé doit répondre par une Message_Data_Response, adressée à un nœud ou en diffusion (voir la Figure 47).



Légende

Anglais	Français
(singlecast or broadcast Slave_Frame)	(Trame_Esclave point à point ou de diffusion)
slave address	adresse de l'esclave
source (master)	(maître) source
destination address	adresse de destination
0..1024 bits of Message_Data	0..1024 bits de Message_Data

Figure 47 – Télégramme de Données de Messagerie

NOTE L'Article 6 spécifie la structure des Données de Messagerie.

5.3.3.2 Message_Data_Request

Le format de Message_Data_Request doit se présenter comme suit (voir la Figure 48):

```
Message_Data_Request::= RECORD
  {
  destination_device    UNSIGNED8            -- adresse 'noeud' ou adresse
                                                'maître' pour une demande du
                                                maître faite à lui-même
  link_control (= MESSAGE_DATA)              -- Message_Data_Request
  source_device         UNSIGNED8            -- adresse 'maître'
  link_data_size        UNSIGNED8            -- =0
  }                                          -- vide
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							

Figure 48 – Format de Message_Data_Request

5.3.3.3 Message_Data_Response

Le format de Message_Data_Response doit se présenter comme suit (voir la Figure 49):

```
Message_Data_Response::= RECORD
  {
  destination_device    UNSIGNED8          -- adresse ‘noeud’ ou
                                              adresse ‘diffusion’
  link_control          Link_Control       -- Message_Data_Response
  source_device         UNSIGNED8          -- adresse ‘noeud’
  link_data_size        UNSIGNED8          -- (0   link_data_size  128)
  ARRAY [link_data_size] OF WORD8          -- contenu des Données de
                                              Messagerie défini par
                                              l'Article 6
  }
```

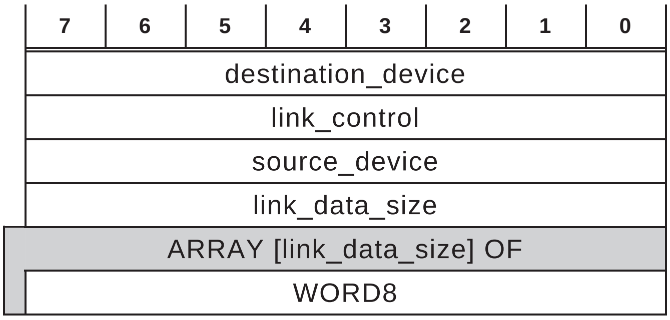
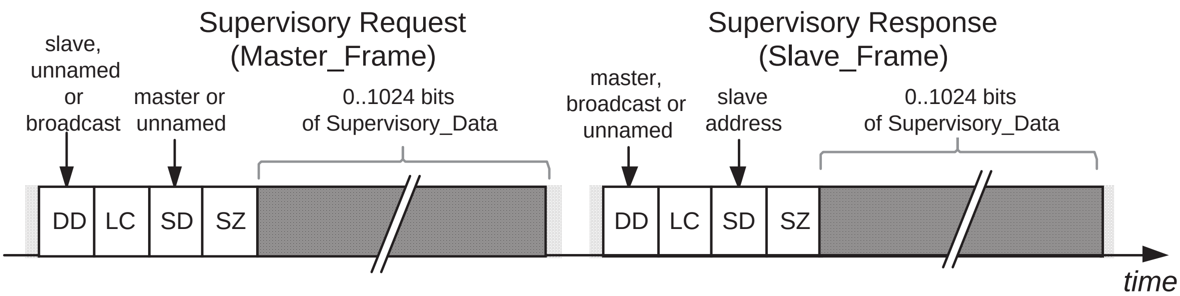


Figure 49 – Format de Message_Data_Response

5.3.4 Données de Supervision

5.3.4.1 Action

Un maître doit demander les Données de Supervision d'un nœud ou en envoyer en utilisant une Supervisory_Data_Request, auquel la source adressée doit répondre par une Supervisory_Data_Response (voir la Figure 50).



Légende

Anglais	Français
Supervisory Request	Demande de supervision
Supervisory Response	Réponse de supervision
slave, unnamed or broadcast	esclave, sans nom ou diffusion
Master or unnamed	maître ou sans nom
0..1024 bits of Supervisory_Data	0..1024 bits de Supervisory_Data
master, broadcast or unnamed	maître, diffusion ou sans nom
slave address	adresse de l'esclave
time	durée

Figure 50 – Télégramme de supervision

Sur le Canal Auxiliaire, le nœud d'Extrémité peut jouer le rôle d'un maître.

5.3.4.2 Formats des télégrammes de supervision

Le format des trames de supervision doit se présenter de la manière suivante:

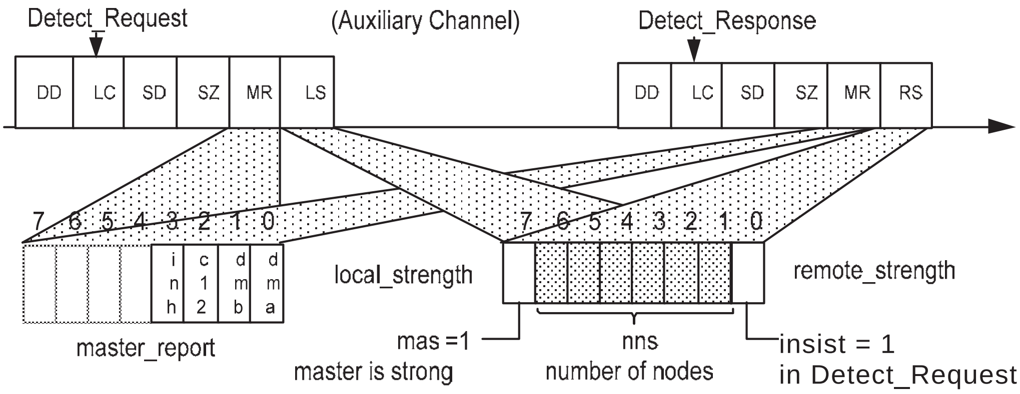
```
Supervisory_Data::= ONE_OF [link_control.mq]
  {
  [MQ]                    Supervisory_Data_Request,
  [SR]                    Supervisory_Data_Response
  }

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
  }
```

5.3.5 Télégramme de détection

5.3.5.1 Action

Un Nœud d'Extrémité doit signaler sa présence à un autre nœud par une Detect_Request, à laquelle l'autre nœud (s'il existe et s'il peut répondre) doit répondre par une Detect_Response.

La Figure 51 illustre un télégramme de détection.



Légende

Anglais	Français
(Auxiliary Channel)	(Canal Auxiliaire)
master is strong	le maître est fort
number of nodes	nombre de nœuds
insist = 1 in Detect_Request	insist = 1 dans Detect_Request

Figure 51 – Télégramme de détection

5.3.5.2 Detect_Request

Le format de Detect_Request doit se présenter de la manière suivante (voir la Figure 52):

```
Detect_Request::= RECORD
  {
  destination_device    UNSIGNED8            -- adresse 'sans_nom'
  link_control          Link_Control         -- Detect_Request
  source_device         UNSIGNED8            -- adresse 'sans_nom'
  link_data_size        UNSIGNED8            -- =2
  master_report         Master_Report        -- voir 5.5.2.5
  local_strength        Composition_Strength    -- copie de LocStr du noeud
                                                requérant (voir 5.5.2.4)
                                             -- 'ins' est mis à '1'.
  }
```

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">destination_device</td></tr>
<tr><td colspan="8">link_control</td></tr>
<tr><td colspan="8">source_device</td></tr>
<tr><td colspan="8">link_data_size</td></tr>
<tr><td colspan="8">master_report</td></tr>
<tr><td>mas</td><td colspan="6">nns</td><td>ins</td></tr>
</table>

Figure 52 – Format de Detect_Request

5.3.5.3 Detect_Response

Le format de Detect_Response doit se présenter de la manière suivante (voir la Figure 53):

```
Detect_Response::= RECORD
  {
  destination_device    UNSIGNED8             -- adresse ‘diffusion’
  link_control          Link_Control          -- Detect_Response
  source_device         UNSIGNED8             -- adresse ‘sans_nom’
  link_data_size        UNSIGNED8             -- =2
  master_report         Master_Report,        -- comme dans Detect_Request
                                                 (pour l’autre composition)
  remote_strength       Composition_Strength    -- RemStr du noeud répondant
                                                qui met ‘ins’ à ‘1’ si sa
                                                composition insiste
                                                (voir 5.5.2.4)
  }
```

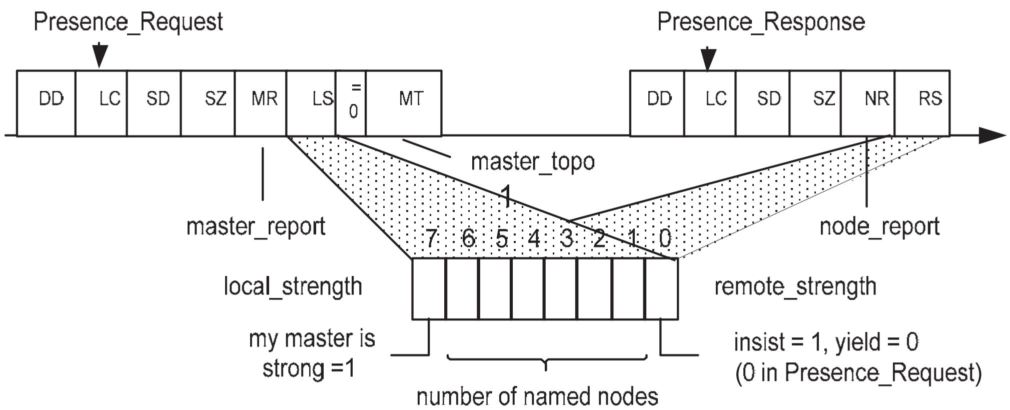
<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">destination_device</td></tr>
<tr><td colspan="8">link_control</td></tr>
<tr><td colspan="8">source_device</td></tr>
<tr><td colspan="8">link_data_size</td></tr>
<tr><td colspan="8">master_report</td></tr>
<tr><td>mas</td><td colspan="6">nns</td><td>ins</td></tr>
</table>

Figure 53 – Format de Detect_Response

5.3.6 Télégramme de présence

5.3.6.1 Action

Le maître doit demander à un Nœud d’Extrémité de signaler sa présence et la présence éventuelle d'une autre composition par une Presence_Request, à laquelle le Nœud d’Extrémité doit répondre par une Presence_Response (voir la Figure 54).



Légende

Anglais	Français
number of named nodes	nombre de nœuds nommés
my master is strong = 1	mon maître est fort = 1

Figure 54 – Télégramme de présence

5.3.6.2 Presence_Request

Le format de Presence_Request doit se présenter de la manière suivante (voir la Figure 55):

```
Presence_Request::= RECORD
  {
  destination_device    UNSIGNED8             -- adresse du Noeud d'Extrémité
  link_control          Link_Control          -- Presence_Request
  source_device         UNSIGNED8             -- adresse 'maître'
  link_data_size        UNSIGNED8             -- = 4
  master_report         Master_Report         -- voir 5.5.2.5
  local_strength        Composition_Strength   -- copie de LocStr du
                                               maître, 'ins' est à '0'.
  reserved1             WORD4 (=0)            -- réservé, = 0
  master_topo           Master_Topo           -- voir 5.5.2.7
  }
```

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">destination_device</td></tr>
<tr><td colspan="8">link_control</td></tr>
<tr><td colspan="8">source_device</td></tr>
<tr><td colspan="8">link_data_size</td></tr>
<tr><td colspan="8">master_report</td></tr>
<tr><td>mas</td><td colspan="6">nns</td><td>ins</td></tr>
<tr><td colspan="4">reserved1</td><td colspan="4"></td></tr>
<tr><td colspan="8">master_topo</td></tr>
</table>

Figure 55 – Format de Presence_Request

5.3.6.3 Presence_Response

Le format de Presence_Response doit se présenter de la manière suivante (voir la Figure 56):

```
Presence_Response::= RECORD
  {
  destination_device    UNSIGNED8             -- adresse 'diffusion'
  link_control          Link_Control          -- Presence_Response
  source_device         UNSIGNED8             -- adresse du Noeud d'Extrémité
  link_data_size        UNSIGNED8             -- =2
  node_report           Node_Report           -- voir 5.5.2.2
  remote_strength       Composition_Strength   -- copie de RemStr du
                                               Noeud d'Extrémité
                                               'ins' = '1' indique que
                                               l'autre composition insiste.
```

```
}
```

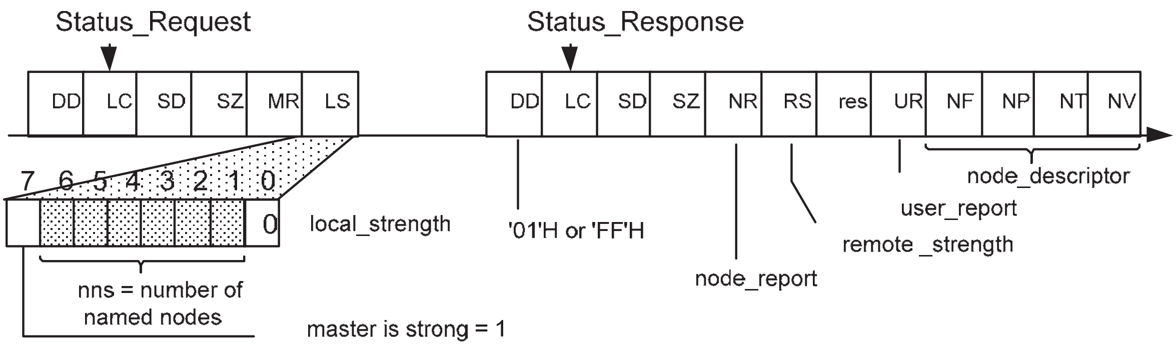
7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							
node_report							
mas	nns						ins

Figure 56 – Format de Presence_Response

5.3.7 Télégramme de statut

5.3.7.1 Action

Le maître doit demander le statut d'un nœud par une Status_Request, à laquelle l'esclave doit répondre par une Status_Response (voir la Figure 57).



Légende

Anglais	Français
number of named nodes	nombre de nœuds nommés
or	ou

Figure 57 – Télégramme de statut

5.3.7.2 Status_Request

Le format de Status_Request doit se présenter de la manière suivante (voir la Figure 58):

```
Status_Request::= RECORD
  {
  destination_device    UNSIGNED8             -- adresse 'noeud'
  link_control          Link_Control          -- Status_Request
  source_device         UNSIGNED8             -- adresse 'maître'
  link_data_size        UNSIGNED8             -- = 2
  master_report         Master_Report         -- voir 5.5.2.5
  local_strength        Composition_Strength    -- LocStr du maître, 'ins'
                                                est à 0
  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							
master_report							
mas	nns						ins

Figure 58 – Format de Status_Request

5.3.7.3 Status_Response

Le format de Status_Response doit se présenter de la manière suivante (voir la Figure 59):

```
Status_Response::= RECORD
  {
  destination_device    UNSIGNED8             -- adresse 'maître' ou
                                                 'diffusion'
  link_control          Link_Control          -- Status_Response
  source_device         UNSIGNED8             -- adresse 'noeud'
  link_data_size        UNSIGNED8             -- = 8
  node_report           Node_Report           -- voir 5.5.2.2
  remote_strength       Composition_Strength     -- composition éloignée
                                                 force du noeud d'extrémité,
                                                 0 pour un noeud
                                                 intermédiaire.
  reserved1             WORD8 (=0)            -- réservé, = 0
  user_report           User_Report           -- voir 5.5.2.3
  node_descriptor       Node_Descriptor       -- voir 5.5.2.1
  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							
node_report							
mas	nns						ins
reserved1							
user_report							
node_frame_size							
reserved1					node_period		
node_type							
node_version							

Figure 59 – Format de Status_Response

5.3.8 Télégramme de mise en position intermédiaire

5.3.8.1 Action

Le maître doit demander à un esclave de mettre ses commutateurs en position intermédiaire (Intermediate_Setting) en envoyant une SetInt_Request que l'esclave doit acquitter par une SetInt_Response (voir la Figure 60).

Figure 60 – Télégramme de mise en position intermédiaire

5.3.8.2 SetInt_Request

Le format de SetInt_Request doit se présenter de la manière suivante (voir la Figure 61):

```
SetInt_Request::= RECORD
  {
  destination_device    UNSIGNED8          -- adresse ‘noeud’
  link_control          Link_Control       -- SetInt_Request
  source_device         UNSIGNED8          -- adresse ‘maître’
  link_data_size        UNSIGNED8          -- =0
  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							

Figure 61 – Format de SetInt_Request

5.3.8.3 SetInt_Response

Le format de SetInt_Response doit se présenter de la manière suivante (voir la Figure 62):

```
SetInt_Response::= RECORD
  {
  destination_device    UNSIGNED8          -- adresse ‘maître’
  link_control          Link_Control       -- SetInt_Response
  source_device         UNSIGNED8          -- adresse ‘noeud’
  link_data_size        UNSIGNED8          -- =0
  }
```

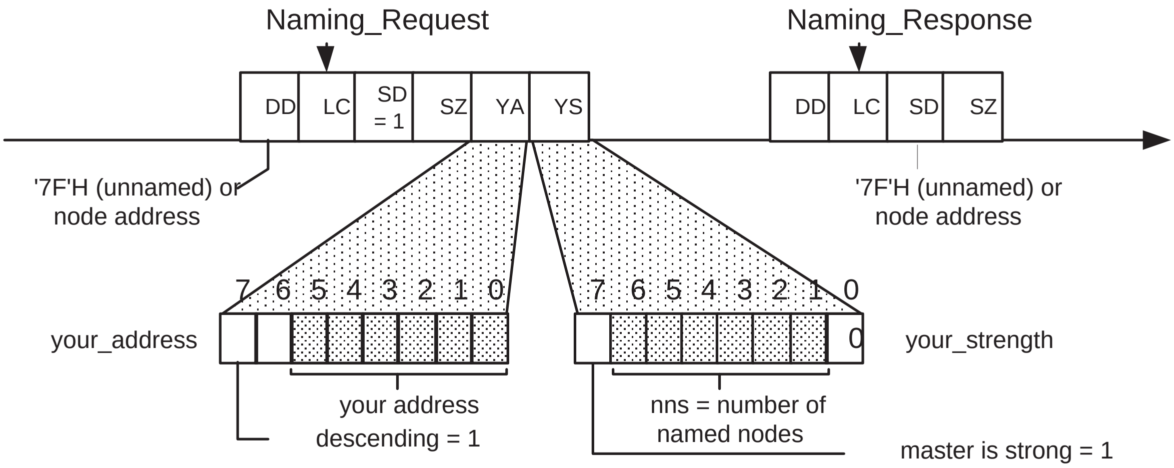
7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							

Figure 62 – Format de SetInt_Response

5.3.9 Télégramme de nomination

5.3.9.1 Action

Le maître doit communiquer son adresse allouée et sa force à un esclave en envoyant une Naming_Request, que l'esclave doit acquitter par une Naming_Response (voir la Figure 63).



Légende

Anglais	Français
7F'H (unnamed) or node address	7F'H (sans nom) ou adresse de nœud
your address descending = 1	votre adresse en séquence descendante = 1
number of named nodes	nombre de nœuds nommés
master is strong = 1	le maître est fort = 1

Figure 63 – Télégramme de nomination

5.3.9.2 Naming_Request

Le format de Naming_Request doit se présenter de la manière suivante (voir la Figure 64):

```
Naming_Request::= RECORD
  {
  destination_device    UNSIGNED8            -- adresse 'noeud' ou adresse
                                                'sans_nom'
  link_control          Link_Control         -- Naming_Request
  source_device         UNSIGNED8            -- adresse 'maître'
  link_data_size        UNSIGNED8            -- = 2
  dir1                  BOOLEAN1             -- '1' si le noeud est nommé en
                                                direction ascendante
  rsv1                  WORD1                -- réservé, = 0
  your_address          UNSIGNED6            -- adresse donnée par le maître
                                                à ce noeud
  your_strength         Composition_Strength    -- attribué par le maître,
                                                'ins' = 0
  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							
dir1	rsv1	your_address					
mas	nns						ins

Figure 64 – Format de Naming_Request

5.3.9.3 Naming_Response

Le format de Naming_Response doit se présenter de la manière suivante (voir la Figure 65):

```
Naming_Response::= RECORD
  {
  destination_device    UNSIGNED8           -- adresse ‘maître’ ou adresse
                                               ‘sans_nom’
  link_control          Link_Control        -- Naming_Request
  source_device         UNSIGNED8           -- adresse ‘noeud’ ou adresse
                                               ‘sans_nom’(voir 0)
  link_data_size        UNSIGNED8           -- =0
  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							

Figure 65 – Format de Naming_Response

5.3.10 Télégramme de dénommage

5.3.10.1 Action

Le maître doit demander à tous les esclaves de se dénommer en diffusant une Unname_Request, à laquelle les esclaves ne doivent pas répondre, comme l'illustre la Figure 66 (il n’y a pas d’Unname_Response).

Figure 66 – Télégramme de dénommage

5.3.10.2 Unname_Request

Le format de Unname_Request doit se présenter de la manière suivante (voir la Figure 67):

```
Unname_Request::= RECORD
  {
  destination_device    UNSIGNED8          -- adresse ‘diffusion’
  link_control          Link_Control       -- Unname_Request
  source_device         UNSIGNED8          -- adresse ‘maître’
  link_data_size        UNSIGNED8          -- =0
  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							

Figure 67 – Format de Unname_Request

5.3.11 Télégramme de mise en position terminale

5.3.11.1 Action

Le maître doit demander à un esclave de basculer sur End_Setting et d'accepter une nouvelle force de composition en envoyant une SetEnd_Request, que l'esclave doit acquitter par une SetEnd_Response (voir la Figure 68).

Set End_Request	End_Request activée
Set End_Response	End_Response activée

Figure 68 – Télégramme de mise en position terminale

5.3.11.2 SetEnd_Request

Le format de SetEnd_Request doit se présenter de la manière suivante (voir la Figure 69):

```
SetEnd_Request::= RECORD
  {
  destination_device    UNSIGNED8          -- adresse ‘noeud’
  link_control          Link_Control       -- SetEnd_Request
  source_device         UNSIGNED8          -- adresse ‘maître’
  link_data_size        UNSIGNED8          -- = 2
  reserved1             WORD8 (=0)         -- = 0
  local_strength        Composition_Strength   -- LocStr vue par le maître
                                               (voir 5.5.2.4)
  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							
reserved1							
mas	nns						ins

Figure 69 – Format de SetEnd_Request

5.3.11.3 SetEnd_Response

Le format de SetEnd_Response doit se présenter de la manière suivante (voir la Figure 70):

```
SetEnd_Response::= RECORD
  {
  destination_device     UNSIGNED8          -- adresse 'maître'
  link_control           Link_Control       -- SetEnd_Response
  source_device          UNSIGNED8          -- adresse 'noeud'
  link_data_size         UNSIGNED8          -- = 0
  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							

Figure 70 – Format de SetEnd_Response

5.3.12 Télégramme de Topographie

5.3.12.1 Action

Le maître doit communiquer sa Topographie à un esclave en envoyant une Topography_Request, que l'esclave doit acquitter par une Topography_Response (voir la Figure 71):

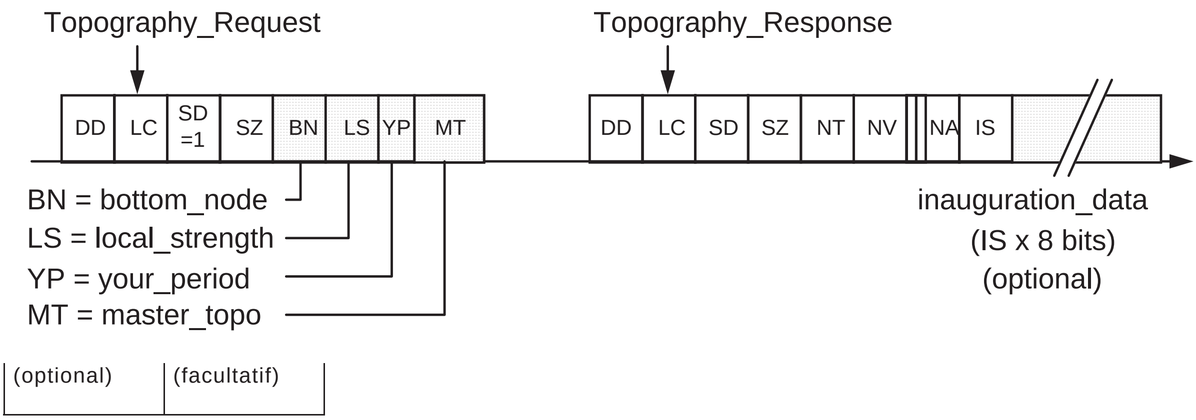


Figure 71 – Télégramme de topographie

5.3.12.2 Topography_Request

Le format de Topography_Request doit se présenter de la manière suivante (voir la Figure 72):

```
Topography_Request::= RECORD
  {
  destination_device    UNSIGNED8             -- adresse ‘noeud’ (qui peut
                                                 être l'adresse du ‘maître’)
  link_control          Link_Control          -- Topography_Request
  source_device         UNSIGNED8             -- adresse ‘maître’
  link_data_size        UNSIGNED8             -- = 4
  bottom_node           UNSIGNED8             -- adresse du Noeud d’Extrémité
                                                 dans la Direction_1 vue du
                                                 maître
  local_strength        Composition_Strength     -- Force de composition vue
                                                 par le maître
                                                 (local_strength.ins = 0)
  your_period           UNSIGNED4             -- Période Individuelle
                                                 assignée (voir 5.5.2.1
                                                 et 5.4.2)
  master_topo           Master_Topo           -- voir 5.5.2.7
  }
```

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">destination_device</td></tr>
<tr><td colspan="8">link_control</td></tr>
<tr><td colspan="8">source_device</td></tr>
<tr><td colspan="8">link_data_size</td></tr>
<tr><td colspan="8">bottom_node</td></tr>
<tr><td>mas</td><td colspan="6">nns</td><td>ins</td></tr>
<tr><td colspan="4">your_period</td><td colspan="4" rowspan="2">master_topo</td></tr>
</table>

Figure 72 – Format de Topography_Request

5.3.12.3 Topography_Response

Le format de Topography_Response doit se présenter de la manière suivante (voir la Figure 73):

```
Topography_Response::= RECORD
  {
  destination_device    UNSIGNED8           -- adresse ‘diffusion’
  link_control          Link_Control        -- Topography_Response
  source_device         UNSIGNED8           -- adresse du noeud source (peut
                                               être le maître)
  link_data_size        UNSIGNED8           -- 0 ‘ link_data_size ‘ 128
  node_type             Node_Type           -- première partie du Node_Key
  node_version          Node_Version        -- deuxième partie du Node_Key
  sam                   BOOLEAN1            -- ‘1’ si la direction est
                                               identique à celle du maître
  rsv1                  WORD1               -- réservé, = 0
  node_address          UNSIGNED6           -- adresse du noeud donnée par
                                               l’inauguration
  inaug_data_size       UNSIGNED8           -- 0 < taille des données
                                               d’inauguration ≤ 124 octets)
  inauguration_data     ARRAY [inaug_data_size] OF WORD8
                                               données d’inauguration
                                               définies par l’application

  }
```

7	6	5	4	3	2	1	0
destination_device							
link_control							
source_device							
link_data_size							
node_type							
node_version							
sam	rsv1	node_address					
inaug_data_size							
ARRAY [inaug_data_size] OF							
WORD8							

Figure 73 – Format de Topography_Response

NOTE La redondance de ‘inaug_data_size’ avec ‘link_data_size’ est intentionnelle et permet de contrôler la plausibilité.

5.4 Attribution du support

5.4.1 Organisation

Les spécifications suivantes s'appliquent au fonctionnement normal, lorsqu'un seul maître est établi et que le bus est capable de transporter les données d’application. Le fonctionnement normal commence dès la fin de l’inauguration et se termine lors d'un changement de composition.

NOTE Le paragraphe 5.5 définit la sélection du maître parmi plusieurs nœuds. La sélection du maître ne fait pas partie de l'attribution du support définie ci-dessous.

5.4.1.1 Période de Base

5.4.1.1.1 Structure de la Période de Base

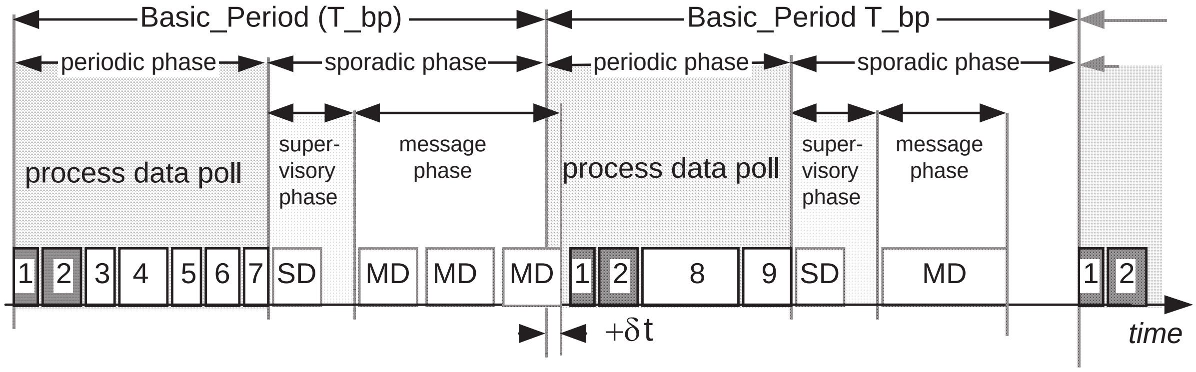
Le maître doit diviser l'activité du bus en périodes, appelées Périodes de Base.

Une Période de Base doit être divisée en deux phases (voir la Figure 74):

a) une Phase Périodique permettant de transmettre les données périodiques, et

b) une Phase Apériodique permettant de transmettre des:

- Données de Supervision; et/ou
- Données de Messagerie.



Légende

Anglais	Français
periodic phase	phase périodique
sporadic phase	phase apériodique
process data poll	interrogation de données de processus
supervisory phase	phase de supervision
message phase	phase de messagerie
time	durée

Figure 74 – Structure de la Période de Base

5.4.1.1.2 Durée de la Période de Base

La durée de la Période de Base doit être T_bp = 25,0 ms ± 1,0 ms.

Le maître peut ne pas émettre de Message_Data_Request ou de Supervisory_Data_Request après le début de la Période de Base, tant que la Phase Périodique n'est pas terminée.

NOTE Cette restriction signifie que le début de la Phase Périodique peut souffrir un délai δt égal à la durée de transmission la plus longue possible de la trame Données de Messagerie ou Données de Supervision y compris l'interrogation, soit environ 2,0 ms. La prochaine Période de Base devrait démarrer en cadence normale s'il n'y a pas d'autres données apériodiques à transmettre.

5.4.2 Phase Périodique

5.4.2.1 Période Individuelle

L'intervalle entre deux interrogations consécutives d'un même nœud est appelé Période Individuelle, T_ip.

La Période Individuelle doit être un multiple de la Période de Base T_bp, de sorte que T_ip = (2n × T_bp).

NOTE La Période Individuelle est définie par l'application pour chaque nœud. Chaque nœud annonce sa Période Individuelle désirée (node_period) et la taille de la Trame-Esclave (node_frame_size) voulue dans son Node_Descriptor (voir 5.5.2.1) lors de l'inauguration.

La Période Individuelle la plus longue d'un nœud sur le bus définit la Macro_Period.

5.4.2.2 Liste Périodique

La Liste Périodique est la liste de tous les nœuds interrogés pendant chaque Période de Base d'une Macro_Period. Elle définit également le temps qu'il reste pour la Phase Apériodique de chaque Période de Base.

Lors de l'inauguration, le maître doit configurer la Liste Périodique sur la base de la Période Individuelle réclamée par chaque nœud (Node_Period) et la taille des Données de Processus (Node_Frame_Size) annoncée par chaque nœud lors de l'Inauguration.

Le maître doit distribuer les interrogations équitablement sur les Périodes de Base, de manière à laisser 40 % de chaque Période de Base pour la Phase apériodique.

Si la Phase Périodique occupe plus de 60 % de la Période de Base, les Périodes Individuelles des nœuds avec la période la plus longue doivent être doublées jusqu'à ce que la Phase Périodique occupe moins de 60 % de la Période de Base, moyennée sur la Macro_Period.

Si cela n'est pas suffisant, la période des nœuds avec la Node_Period la plus longue en second doit être doublée et ainsi de suite, jusqu'à ce que la période des nœuds dont la période est la plus courte soit doublée, si cela est nécessaire.

La Période Individuelle choisie pour chaque nœud doit être communiquée à chaque nœud dans la Topography_Request comme your_period.

5.4.2.3 Interrogation des Nœuds d'Extrémité

Le maître doit interroger un Nœud d'Extrémité dans une Période de Base, et l'autre Nœud d'Extrémité dans la période suivante, en envoyant une trame de demande de présence (Presence_Request), à laquelle le Nœud d'Extrémité doit répondre en diffusant une trame de réponse de présence (Presence_Response).

NOTE Le télégramme de présence permet à tous les nœuds de contrôler l'intégrité du bus, et au maître de détecter la présence d'une autre composition.

5.4.2.4 Conditions d'erreur et traitement

Le maître ne doit pas enlever de sa Liste Périodique un nœud qui ne répond plus. Il doit continuer à l'interroger jusqu'à l'inauguration suivante ou jusqu'à ce que ce nœud réintègre le bus.

NOTE 1 Des nœuds défaillants peuvent uniquement être retirés de la composition lors d'une nouvelle inauguration.

Un nœud doit être capable de signaler à son application la disparition d'un nœud auquel il est abonné pour les Données de Processus, et qui n'a pas répondu à trois interrogations consécutives. Il doit être également capable de signaler la réintégration du nœud dans le cas où il réapparait.

NOTE 2 La supervision du contrôle du temps des Données de Processus permet le contrôle des nœuds manquants.

En fonctionnement normal, le comportement d'un nœud qui n'observe plus le trafic prévu (par exemple Nœuds d'Extrémité manquants, maître manquant ou plus d'interrogation) doit être celui défini en 5.5.4.9.3.

5.4.3 Phase apériodique

5.4.3.1 Annonce d'événements

Un nœud doit demander une émission apériodique en attribuant la valeur 1 à ‘A_bit’ ou ‘C_bit’ dans son Process_Data_Response ou Message_Data_Response.

Lorsque plusieurs nœuds annoncent une émission apériodique pendant la Phase Périodique, le maître doit traiter ces demandes chronologiquement de manière à traiter toutes les autres demandes avant de traiter de nouveau le même nœud.

Toutes les demandes Données de Supervision (‘C_bit‘) doivent être traitées avant les demandes Données de Messagerie (‘A_bit‘).

5.4.3.2 Liste de Messages

Le maître doit insérer dans sa Message_List les adresses des nœuds qui ont activé le ‘A_bit’ de l’une des trames Process_Data ou Message_Data précédentes.

Le maître doit interroger un nœud qui signale un changement par une Message_Data_Request et retirer ce nœud de la Message_List lorsqu'il interroge ce nœud pour les Données de Messagerie, sauf si Message_Data_Response a aussi un ‘A_bit‘ activé.

5.4.3.3 Liste de Supervision

Le maître doit insérer dans sa Supervisory_List (liste de supervision) les adresses des nœuds qui ont activé le ‘C_bit‘ dans l'une des trames Données de Processus ou Données de Messagerie précédentes.

Le maître doit interroger un nœud qui signale un changement en envoyant une Status_Request et enlever ce nœud de la Supervisory_List lorsqu'il reçoit sa Status_Response.

NOTE Normalement, cette liste est vide. Elle contient l'adresse du Nœud d’Extrémité en cas de prolongement du bus, ainsi que les adresses des nœuds qui changent de descripteur ou qui annoncent un changement de la demande de veille (mettre en veille ou annuler la veille).

5.4.3.4 Exploration en arrière plan (en option)

Le maître peut interroger les Données de Messagerie ou le statut des nœuds qui ne sont pas dans sa Message_List ou sa Supervisory_List.

NOTE Un nœud qui n'envoie pas de Données de Processus mais qui peut envoyer des Données de Messagerie est exploré en arrière-plan.

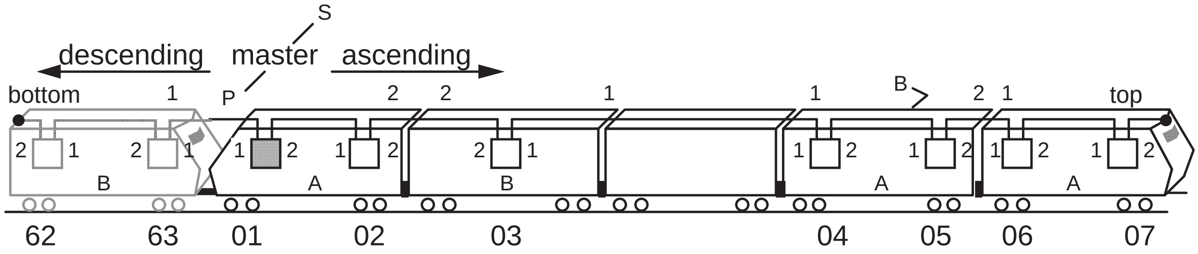
5.5 Inauguration

5.5.1 Généralités

5.5.1.1 Attribution d'adresse

La procédure d’inauguration doit attribuer une adresse à chaque nœud (voir la Figure 75):

- les nœuds dans la Direction_1 par rapport au maître sont numérotés en séquence descendante, en commençant par 63, le nœud nommé en dernier étant le nœud du bas;
- les nœuds dans la Direction_2 par rapport au maître sont numérotés en séquence ascendante en commençant par 02, le nœud nommé en dernier étant le nœud du haut.



Légende

Anglais	Français
descending	descendant
master	maître
ascending	ascendant
bottom	bas
top	haut

Figure 75 – Numérotation de la position des nœuds

5.5.1.2 Classement des nœuds

L'application peut adopter différentes stratégies pour l'élection du maître en classant un nœud soit:

- comme un nœud fort, soit
- comme un nœud faible ou
- comme un nœud esclave.

5.5.1.3 Nœud fort

Un nœud fort est promu par l'application pour devenir le maître.

Un nœud fort qui exerce la maîtrise est appelé maître fort.

Un nœud fort peut être déclassé par l'application au niveau de nœud faible. S'il était maître fort, il informe tous les autres nœuds de ce déclassement et continue à commander le bus en tant que maître faible jusqu'à ce qu'un nœud fort soit promu.

NOTE 1 Normalement, l'application ne nomme qu'un seul nœud fort dans une composition. En présence de plusieurs nœuds forts sur le bus, la procédure d'inauguration provoque la segmentation du bus en autant de segments indépendants qu'il y a de nœuds forts, puisqu'il ne peut y avoir qu'un seul maître par segment.

NOTE 2 Un nœud fort permet de lier l'état de maître à certaines fonctions de l'application (avec le véhicule pilote, par exemple). Une telle liaison est nécessaire pour inclure les Données de Processus dans les trames de Process_Data_Request (5.3.1).

5.5.1.3.1 Nœud faible

Un nœud faible est un nœud auquel l'application permet de devenir maître.

Un nœud faible exerçant la maîtrise est appelé un maître faible.

Normalement, l'application assigne plusieurs nœuds ou tous les nœuds comme nœuds faibles dans une composition.

Dans une composition qui n'a pas de nœud fort, une résolution de conflit qui traite tous les nœuds sur un pied d'égalité assure qu'un seul nœud faible devient maître faible et les autres nœuds deviennent esclaves.

Si un maître faible détecte la présence d'un nœud fort ou d'un autre maître faible qui commande un plus grand nombre d'esclaves, il est déchu pour devenir un esclave de l'autre maître.

Un nœud faible que l'application promeut nœud fort devient un maître fort et inaugure le bus s'il n'est pas déjà maître.

NOTE Les nœuds faibles sont autorisés à faire fonctionner le bus sans une commande explicite de l'application. Les nœuds faibles peuvent pallier la défaillance d'un maître en procédant à l'inauguration et en nommant un autre nœud (faible) comme maître.

5.5.1.3.2 Nœud esclave

Un nœud esclave est un nœud auquel l'application ne permet pas d'exercer la maîtrise de bus.

NOTE Par conséquent, les nœuds esclaves ne contribuent pas à la reprise en cas de défaillance. Ce mode sert pour l'essai ou pour des configurations comportant des nœuds forts.

5.5.2 Descripteurs

Les structures de données suivantes sont utilisées dans les trames de Données de Supervision et dans les messages de gestion de bus. A ce titre, elles sont définies dans la notation de transfert. Pour l'interface de couche de liaison, les types de données C correspondants sont définis en 5.6.4.

5.5.2.1 Descripteur de Nœud

Chaque nœud doit mettre en œuvre un Node_Descriptor pour identifier ses caractéristiques.

Le Node_Descriptor doit être représenté par la structure de données suivante (voir la Figure 76):

```
Node_Descriptor::=      RECORD
  {
  node_frame_size       UNSIGNED8,          -- 'link_data_size' de
                                               Process_Data_Response de ce
                                               noeud
  reserved1             WORD5 (=0)          -- réservé, = 0
  node_period           UNSIGNED3           -- Période Individuelle demandée
                                               par le noeud, exprimée par
                                               le multiple 2n de la période
                                               de base 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           -- première partie du Node_Key
  node_version          UNSIGNED8           -- deuxième partie du Node_Key.
  }
```

7	6	5	4	3	2	1	0
node_frame_size							
reserved1					node_period		
node_type							
node_version							

Figure 76 – Format de Node_Descriptor

NOTE 1 Les Node_Descriptor d'une application donnée font l'objet d'un accord entre les fabricants et les utilisateurs.

NOTE 2 Le Node_Descriptor est utilisé dans Naming_Response, Status_Response et Topography_Response, et comme variable interne.

NOTE 3 Le Node_Descriptor UIC pour les voitures internationales est spécifié dans le CODE UIC 556.

NOTE 4 L'application peut inclure le Node_Key dans les deux premiers octets de chaque trame de Données de Processus pour assurer un niveau de protection supérieur contre la falsification (5.6.2.2).

5.5.2.2 Node_Report

Chaque nœud doit mettre en œuvre un Node_Report pour signaler les perturbations sur la ligne et les changements de la composition.

Le Node_Report doit être représenté par la structure suivante (voir la Figure 77):

```
Node_Report::= BITSET8
  {
  da1                 (0)                 -- Line_A1 perturbée, copie de
                                             DA1 de 4.7.2.4
  da2                 (1)                 -- Line_A2 perturbée, copie de
                                             DA2 de 4.7.2.4
  dB1                 (2)                 -- Line_B1 perturbée, copie
                                             de dB1 de 4.7.2.4
  dB2                 (3)                 -- Line_B2 perturbée, copie
                                             de dB2 de 4.7.2.4
  int                 (4)                 -- activé si le noeud est en
                                             position
                                             Intermediate_Setting,
                                             désactivé s'il est en
                                             position End_Setting.
  dsc                 (5)                 -- activé si le Node_Descriptor
                                             a été changé,
                                             désactivé quand le noeud
                                             reçoit une nouvelle
                                             Topographie.
  slp                 (6)                 -- activé si le noeud souhaite
                                             passer à l'état de veille,
                                             désactivé quand cette
                                             demande est retirée.
  sam                 (7)                 -- activé si le noeud a la même
                                             orientation que son maître,
                                             désactivé s'il a la
                                             direction opposée.
 }
```

7	6	5	4	3	2	1	0
da1	da2	db1	db2	int	dsc	slp	sam

Figure 77 – Format de Node_Report

5.5.2.3 User_Report

Chaque nœud doit mettre en œuvre un User_Report pour émettre un octet spécifique à l'application vers d'autres applications (pour signaler des perturbations propres à une application, par exemple).

User_Report doit être représenté par un octet (voir la Figure 78):

```
User_Report::= WORD8                        -- défini par l'application
```

7	6	5	4	3	2	1	0
ur7	ur6	ur5	ur4	ur3	ur2	ur1	ur0

Figure 78 – Format de User_Report

NOTE Une application peut activer les différents bits de User_Report en utilisant les services de gestion de la couche de liaison.

5.5.2.4 Force de la Composition

Chaque nœud doit mettre en œuvre une variable LocStr (Local Composition Strength) pour indiquer le nombre de nœuds dans la composition et si son maître est fort ou faible.

Chaque nœud doit mettre en œuvre une variable Remote Composition Strength pour chaque Canal Auxiliaire, RemStr (1) et RemStr (2), afin d'indiquer la force d'une composition éloignée dans la Direction_1 ou la Direction_2.

NOTE Ces variables ne sont utilisées que si le canal correspondant est activé. Un nœud intermédiaire n'en a pas besoin.

La force de la composition doit être représentée par la structure suivante (voir la Figure 79):

```
Composition_Strength::= RECORD
  {
  mas                  BOOLEAN1,           -- activé si le maître de cette
                                              composition est fort,
                                              désactivé si le maître est
                                              faible.
  nns                  UNSIGNED6,          -- nombre de noeuds nommés dans
                                              la composition
  ins                  BOOLEAN1,           -- activé si dans un conflit
                                              cette composition insiste
                                              sur sa priorité,
                                              désactivé si elle cède.
  }
```

7	6	5	4	3	2	1	0
mas	nns						ins

Figure 79 – Format de Composition_Strength

Pour simplifier les comparaisons de force, la force de la composition locale ou éloignée doit être calculée comme un octet non signé dont la valeur est:

RemStr ou LocStr = (mas × 128) + 2 × nns + ins;

EXEMPLES

RemStr = 0	aucun autre nœud détecté;
RemStr = 1	un nœud sans nom a été trouvé;
RemStr = 2	un seul maître faible cédant;
RemStr = 3	un seul maître faible insistant;
RemStr = 4	un maître faible cédant avec un nœud;
RemStr = 13	un maître faible insistant avec six nœuds;
RemStr > 128	une composition nommée par un maître fort.

5.5.2.5 Master_Report

Chaque nœud doit mettre en œuvre le Master_Report pour signaler les perturbations de redondance et pour permettre l'identification de la direction de la perturbation.

Master_Report doit être représenté par la structure suivante (voir la Figure 80):

```
Master_Report::=       BITSET8
  {
  rsv1 (=0)                              -- réservé, =0
  rsv2 (=0)                              -- réservé, =0
  rsv3 (=0)                              -- réservé, =0
  rsv4 (=0)                              -- réservé, =0
  inh                                    -- activé si un noeud de cette
                                            composition empêche
                                            l'inauguration
  c12                                    -- activé si cette trame est
                                            envoyée dans la Direction_2
                                            par rapport à ce noeud.
  dmb                                    -- activé si la Line_B du
                                            Canal Principal est
                                            perturbée (copie du bit dB1
                                            ou dB2)
  dma                                    -- activé si la Line_A du
                                            Canal Principal est
                                            perturbée (copie du bit dB1
                                            ou dB2)
  }
```

7	6	5	4	3	2	1	0
rsv1	rsv2	rsv3	rsv4	inh	c12	dmb	dma

Figure 80 – Master_Report

5.5.2.6 Topo_Counter

Chaque nœud doit mettre en œuvre le Topo_Counter, qui est un compteur modulo 64 augmenté de 1 ou 2 pour chaque Topographie successive reçue par ce nœud. Ce compteur ne doit pas prendre la valeur zéro, mais passer directement de 63 à 1.

Topo_Report doit être représenté par la structure suivante (voir la Figure 81):

```
Topo_Counter::=        UNSIGNED6          -- 0 = non utilisé
```

7	6	5	4	3	2	1	0
x	x	topo_counter					

Figure 81 – Format de Topo_Counter

NOTE 1 Topo_Counter est utilisé par les Protocoles en Temps Réel pour garantir la cohérence des messages échangés à travers le Bus de Train pendant une inauguration.

Après une mise hors tension temporaire, le nœud doit assurer le changement de Topo_Counter par rapport à la valeur avant la mise hors tension.

NOTE 2 Topo_Counter peut être enregistré dans une mémoire non volatile (utilisée pour les réinsertions rapides). Lors de la remise sous tension, le nœud est en mesure d'obtenir son Topo Counter précédent et de le modifier.

Après une commutation de redondance, le nœud nouvellement actif doit assurer la modification de Topo Counter par rapport à la valeur du nœud précédemment activé.

NOTE 3 Un nœud peut utiliser un Topo_Counter pair, l'autre nœud pouvant utiliser un Topo_Counter impair. Topo_Counter est augmenté de 2 pour chaque Topographie successive qu'il reçoit. Cette méthode permet de s'assurer que Topo_Counter reste respectivement pair ou impair. En cas de commutation de redondance, Topo_Counter change de pair à impair, ou inversement.

5.5.2.7 Master Topo

Chaque nœud doit mettre en œuvre le Master_Topo, qui est un compteur de 12 bits augmenté de 1 pour chaque Topographie successive distribuée par le maître.

Master_Topo doit être représenté par la structure suivante (voir la Figure 82):

```
Master_Topo::=         UNSIGNED12
```

11	10	9	8	7	6	5	4	3	2	1	0
master_topo											

Figure 82 – Format de Master_Topo

Quand un nœud devient maître pour la première fois, le Master_Topo de 12 bits doit être initialisé de manière aléatoire.

Le maître doit incrémenter le Master_Topo de un chaque fois qu'il distribue la Topographie.

NOTE Ce compteur permet de déconnecter provisoirement les nœuds afin de vérifier qu'ils réintégrent la composition correcte.

5.5.2.8 Inauguration_Counter

Chaque nœud doit mettre en œuvre un Inauguration_Counter de 16 bits pour enregistrer le nombre de fois qu'il a participé à une inauguration, en repassant par l'état de nœud sans nom.

NOTE Ce compteur est utilisé à des fins de diagnostic.

5.5.3 Détection d'autres compositions (informel)

5.5.3.1 Protocole de détection

Les Nœuds d'Extrémité:

- détectent la présence d'un nœud supplémentaire relié à leur(s) extrémité(s) ouvertes; et
- signalent leur propre présence à ce nœud supplémentaire.

Pour cela, un Nœud d'Extrémité envoie une Detect_Request comprenant son LocStr (Local Composition Strength) vers son extrémité ouverte (ou les extrémités ouvertes dans le cas d'un maître seul).

A l'origine, le bit 'insist' activé de cette Detect_Request indique que cette composition insisterait en cas de conflit.

Dans tous les cas, un Nœud d'Extrémité (nommé ou sans nom) répond à une Detect_Request reçue par une Detect_Response (ou une autre Detect_Request) dans le temps imparti de l'intervalle entre les trames (5.2.2.4).

Un Nœud d'Extrémité récepteur compare la force de la composition éloignée (RemStr) par rapport à sa propre force (LocStr):

- si l'autre composition est plus faible que la sienne, le nœud laisse son bit 'insist' activé;
- si l'autre composition est plus forte ou de même force, le nœud désactive son bit 'insist' et cède;
- si les deux compositions sont nommées par un maître fort, il laisse son bit 'insist' activé.

Le nœud qui reçoit une Presence_Request adopte la force indiquée dans Presence_Response.

Un Nœud d'Extrémité signale dans chaque Presence_Response suivante:

a) la présence d'un autre nœud;
b) la force de la composition éloignée;
c) la décision locale de céder ou d'insister en cas de forces identiques.

5.5.3.2 Règles pour éviter les collisions

Comme les Nœuds d'Extrémité de deux compositions différentes envoient des Detect_Request de manière asynchrone, un nœud peut recevoir une trame brouillée ou une autre Detect_Request lorsqu'il attend une Detect_Response. Les cinq règles suivantes réduisent les cas de conflit:

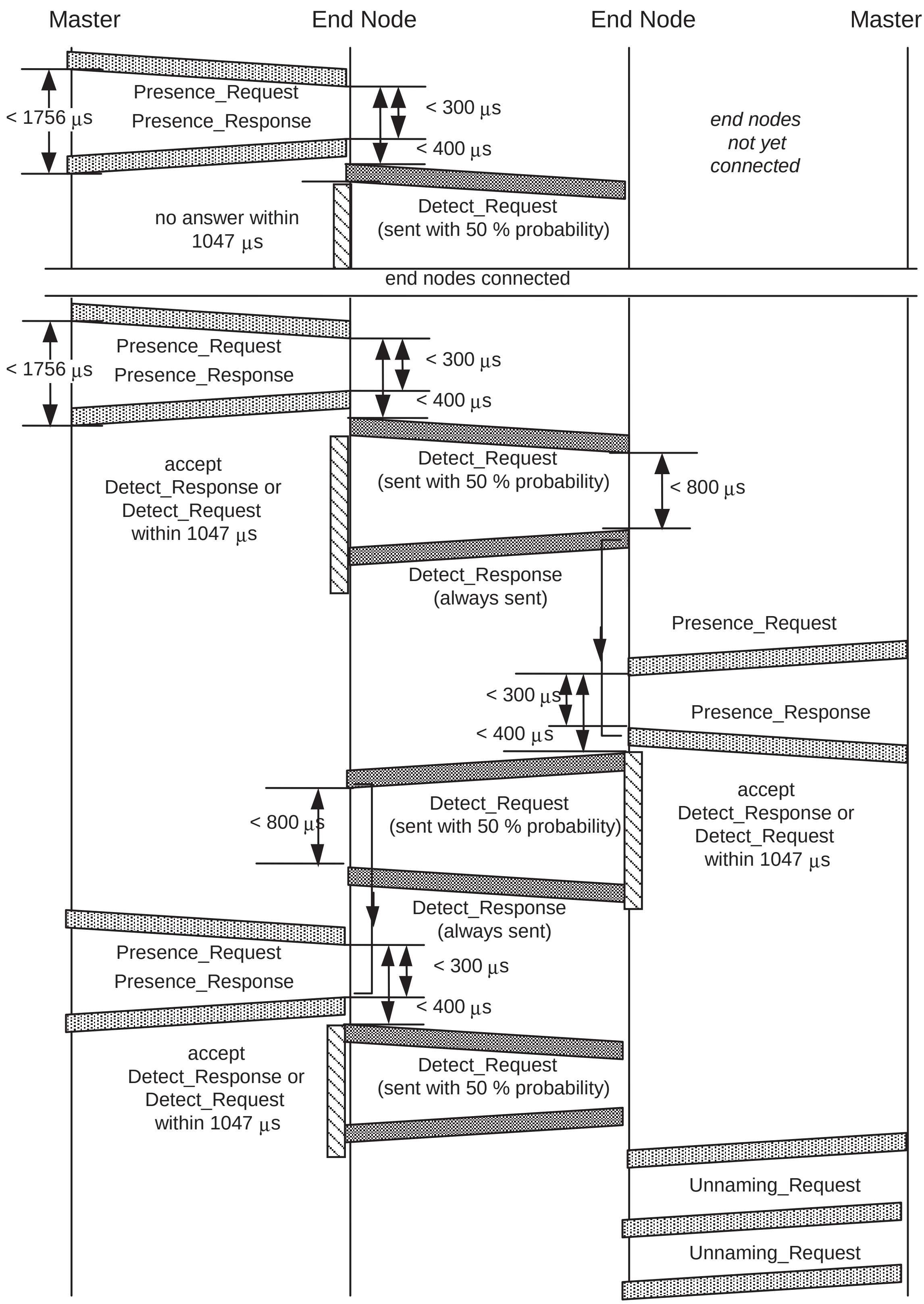
a) un Nœud d'Extrémité qui n'est pas encore en fonctionnement normal doit envoyer une Detect_Request par son Canal Auxiliaire au plus tard 400,0 µs après réception d'une Naming_Request ou une Status_Request provenant de son Canal Principal;
b) un Nœud d'Extrémité en fonctionnement normal doit envoyer une Detect_Request par son Canal Auxiliaire au plus tard 400,0 µs après réception d'une Presence_Request provenant de son Canal Principal, selon une probabilité de 50 %, dans le but d'éviter des collisions répétitives;
c) un maître isolé doit envoyer une Detect_Request par ses deux Canaux Auxiliaires toutes les 29,0 ms au moins, mais toutes les 21,0 ms au plus, en faisant fluctuer le moment d'émission dans un intervalle de ± 4,0 ms autour de 25,0 ms, dans le but d'éviter des collisions répétitives;
d) un Nœud d'Extrémité, après avoir envoyé une Detect_Request, doit ignorer les trames autres que Detect_Response et Detect_Request reçues pendant un temps T_detecting_response = 1,047 ms et doit ignorer les trames autres que Detect_Request et Naming_Request reçues après ce temps;

e) un maître ne doit pas envoyer de Detect_Request à un rythme que le Nœud d'Extrémité ne puisse suivre. Ceci est assuré par le protocole de Status_Request, de Presence_Request et de Naming_Request.

NOTE 1 Un maître qui est également un Nœud d'Extrémité envoie une Presence_Request adressée à lui-même.

NOTE 2 Un maître qui est également un Nœud d'Extrémité envoie une Status_Request adressée à lui-même.

EXEMPLE La Figure 83 montre le chronogramme d'un processus de détection classique.


Master
End Node
End Node
Master
< 1756 µs
Presence_Request
Presence_Response
< 300 µs
< 400 µs
end nodes
not yet
connected
no answer within
1047 µs
Detect_Request
(sent with 50 % probability)
end nodes connected
< 1756 µs
Presence_Request
Presence_Response
< 300 µs
< 400 µs
accept
Detect_Response or
Detect_Request
within 1047 µs
Detect_Request
(sent with 50 % probability)
< 800 µs
Detect_Response
(always sent)
Presence_Request
< 300 µs
< 400 µs
Presence_Response
Detect_Request
(sent with 50 % probability)
< 800 µs
accept
Detect_Response or
Detect_Request
within 1047 µs
Detect_Response
(always sent)
Presence_Request
Presence_Response
< 300 µs
< 400 µs
accept
Detect_Response or
Detect_Request
within 1047 µs
Detect_Request
(sent with 50 % probability)
Unnaming_Request
Unnaming_Request

Légende

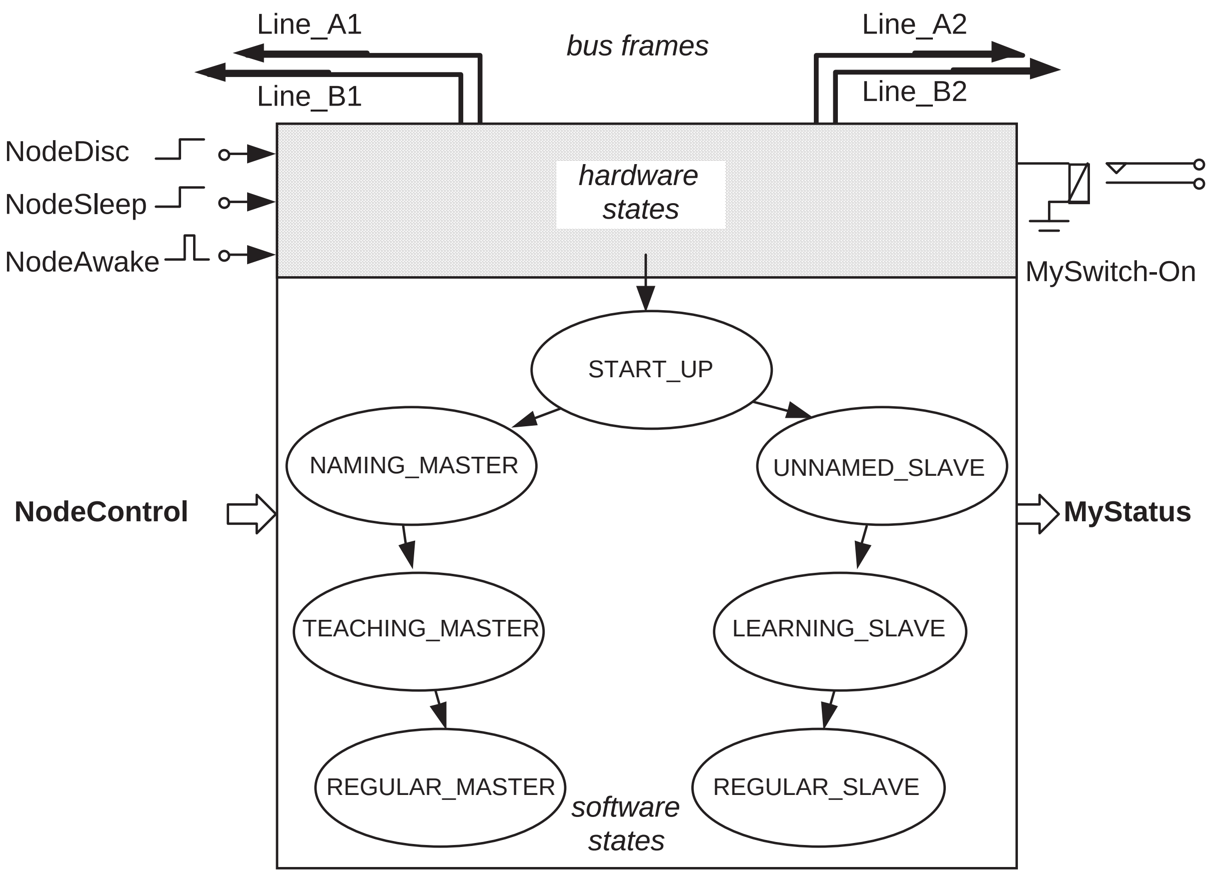
Anglais	Français
accept Detect_Response or Detect_Request within	accepter Detect_Response ou Detect_Request dans les
end nodes not yet connected	nœuds d'extrémité pas encore connectés
end nodes connected	nœuds d’extrémité connectés
no answer within	pas de réponse en
(sent with 50 % probability)	(envoyée avec une probabilité de 50 %)
always sent	toujours envoyés
master	maître
end node	nœud d’extrémité

Figure 83 – Chronogramme du protocole de détection

5.5.4 Diagrammes d'état de l’inauguration

5.5.4.1 Structure des nœuds

Un nœud doit être dans l’un des principaux états présentés à la Figure 84. Ces principaux états sont divisés en états mineurs définis dans les paragraphes suivants.



Légende

Anglais	Français
bus frames	trames du bus
hardware states	états matériels
software states	états logiciels

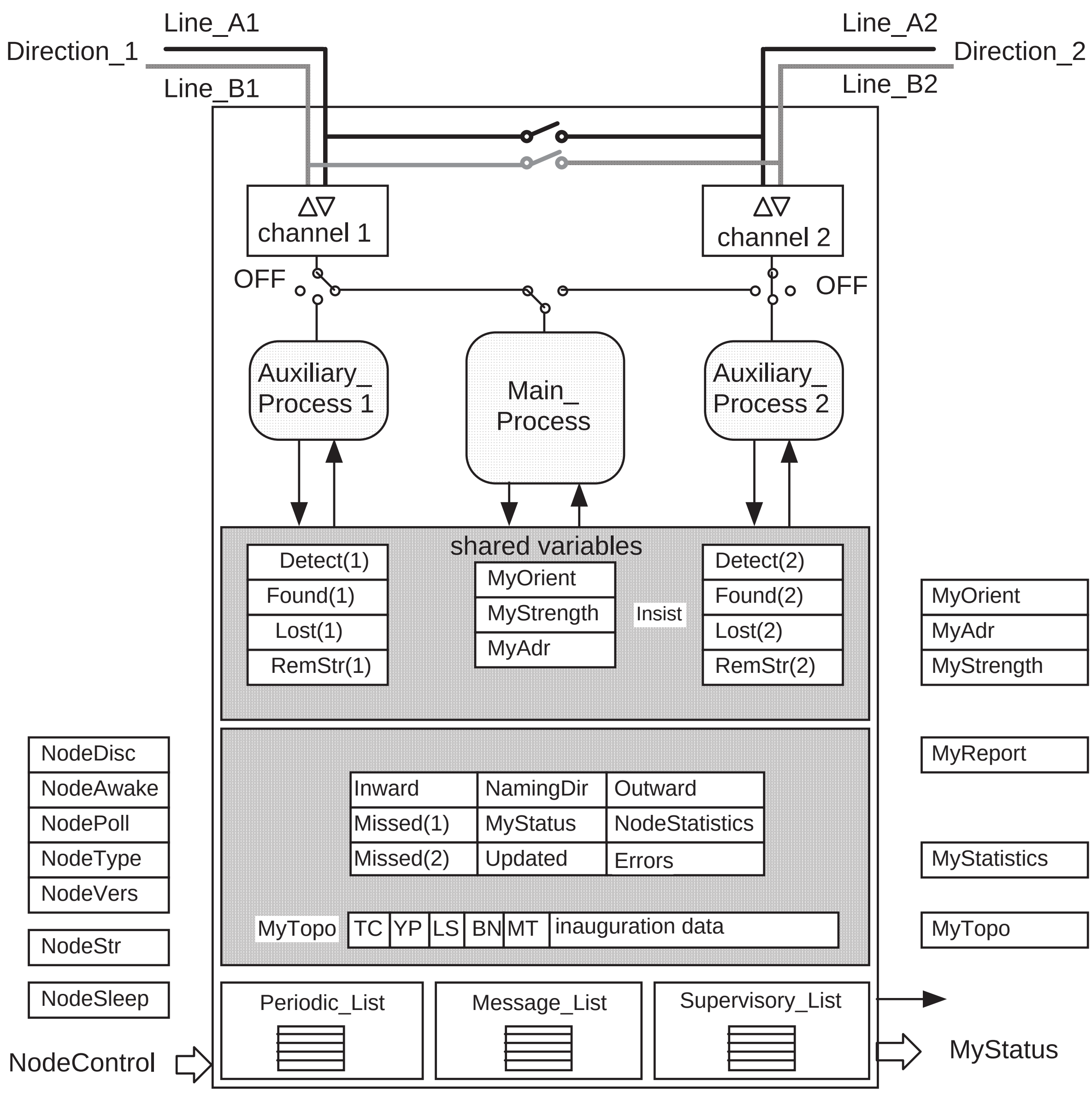
Figure 84 – Principaux états des nœuds et réglages de l'application

Un nœud peut être dans un état de veille à basse puissance pour économiser de l'énergie. Étant donné qu'en mode veille, le nœud n'est pas totalement sous tension, certains états ainsi que leurs transitions doivent être mis en œuvre par le matériel de la MAU. Ils sont donc considérés comme états matériels et les variables de contrôle et d'état correspondantes sont symbolisées par des éléments matériels. Les autres états sont considérés comme états logiciels.

5.5.4.2 Structure du processus d'inauguration

Du point de vue conceptuel et pour les besoins de la présente spécification, l'activité des nœuds est répartie en trois processus (voir la Figure 85):

- un Main_Process (processus principal) activé sur tous les nœuds nommés. Sur un esclave nommé, ce processus est associé à la direction du maître. Sur un maître, le processus est associé dans la direction du premier esclave nommé par le maître;
- deux Auxiliary_Processes (processus auxiliaires) identiques, chacun associé à une direction. Ils ne sont activés que dans les Nœuds d'Extrémité et dans la direction de l'extrémité ouverte du bus.



Légende

Anglais	Français
channel 1	canal 1
channel 2	canal 2
shared variables	variables partagées

Figure 85 – Processus des nœuds (position d'extrémité)

Pour les besoins de la présente spécification, le Main_Process et les Auxiliary_Process sont considérés comme des processus cycliques, c'est-à-dire qu'ils fonctionnent à des intervalles prédéterminés (par défaut, chaque Période de Base) et s'arrêtent d'eux-mêmes avant la fin de l'intervalle. Les Main_Process et Auxiliary_Process d'un nœud fonctionnent en parallèle avec les processus correspondants dans les autres nœuds.

Les processus se contrôlent entre eux (et eux-mêmes):

a) en envoyant des trames sur le bus;

b) en démarrant des temporisations;

c) par des variables communes (interrogées).

Les transitions à l'intérieur d'un processus sont déclenchées par:

d) les variables de commande;
e) les trames reçues sur le bus;
f) les temporisations écoulées.

Les transitions sont conditionnées par les variables communes (interrogées) du même processus ou d'autres processus.

L'état du nœud est rendu visible par la structure ‘MyStatus’.

5.5.4.3 Langage de spécification

La procédure d’inauguration est spécifiée en langage SDL/GR [UIT-T Z.100].

Pour simplifier les diagrammes, les conventions suivantes s'appliquent:

a) en restriction par rapport à SDL, les événements ne sont pas mis en file d'attente, c'est-à-dire qu'un nœud dans un état donné ne prend en compte que les événements qui se produisent lorsque le processus se trouve dans l'état en question;
b) pour simplifier les diagrammes, des instructions ‘IF/ELSE’ sont incluses. Les Booléens prennent les valeurs Oui/Non, 1/0 ou VRAI/FAUX selon le cas;
c) les noms des états ou des macros sont entièrement mis en majuscule. Si une macro ne contient qu’un seul état, ce dernier porte le même nom que la macro mais il n’y a qu’un seul point d’entrée pour la macro;
d) à chaque état peut être associé un temporisateur portant le même nom que l’état (T_named_slave pour l’état NAMED_SLAVE, par exemple), qui est redémarré à chaque entrée dans l’état;
e) une opération retourne à son état d’origine si l’état suivant n’est pas spécifié.

5.5.4.4 Variables des nœuds

5.5.4.4.1 NodeControl

Un nœud doit être contrôlé par la structure de données NodeControl définie dans le Tableau 9.

Tableau 9 – Structure de données NodeControl

Variable	Type	Signification
NodeAwake	BOOLEAN1	commande au nœud de sortir de l'état de veille et lui permet de participer à une inauguration, à condition que NodeSleep et NodeInhibit soient désactivés
NodeDisc	BOOLEAN1	bascule le nœud vers un état passif sur Intermediate_Setting; il est désactivé par le circuit de mise sous tension; en général, il est activé lorsque l'alimentation est insuffisante (moins de 70 % de la valeur nominale, par exemple); ce signal peut être également activé si le nœud subit des dommages irrécupérables.
NodeSetUp	RECORD	commande qui donne au nœud ses données de configuration, et en particulier: NodeDescriptor et NodeStrength (esclave/faible/fort)
NodeInhibit	BOOLEAN1	empêche le nœud de provoquer une inauguration, sauf pour reprise après une défaillance
NodeSleep	BOOLEAN1	demande au nœud de se mettre dans un état de veille à basse puissance sur End_Setting, mais n'empêche pas le nœud de se réveiller s'il détecte une activité. en général, ce signal est activé s'il convient de mettre le bus hors tension (si le chargement de la batterie s'est arrêté depuis plus de 45 min, par exemple)

5.5.4.4.2 MyStatus

Un nœud doit rendre son état disponible à l'application en utilisant la structure de données MyStatus spécifiée dans le Tableau 10.

Tableau 10 – Structure de données MyStatus

Variable	Type	Signification
MyTopo	RECORD	version actuelle de la Topographie reçue dans la dernière distribution; cette variable est une copie de la Topographie (qui peut être incohérente tant que les nœuds n'ont pas reçu la même version).
MyOrient	ANTIVALENT2	indique si le nœud est orienté dans la même direction que le maître ou non. '00'B: erreur, '01'B: direct, '10'B: opposé, '11'B non défini
MyDir	BOOLEAN1	VRAI si le nœud est orienté dans la même direction que le maître, telle que reçue par Naming_Request
MyAdr	UNSIGNED6	reçu de Naming_Request (your_address), ou adresse 'sans nom' pour un nœud sans nom ou adresse 'maître' pour un maître
MyReport	NodeReport	Node_Report dans le format que le nœud enverrait au maître dans une trame Status_Response (5.3.7)
MyStrength	Composition_Strength	force locale de ce nœud comme reçue par Naming_Request sur le Canal Auxiliaire ou dans SetEnd_Request, Status_Request ou Topography_Request sur le Canal Principal
MySwitchOn	BOOLEAN1	'1' si le nœud doit être mis sous tension, '0' bascule le nœud en mode basse puissance
MyStatistics	RECORD	statistiques des trames envoyées et reçues et comptages d'erreur

5.5.4.4.3 Variables partagées

Le Main_Process et les Auxiliary_Processes échangent des données par des variables partagées.

Ces variables sont censées être interrogées par les processus respectifs, c'est-à-dire qu'il n'existe pas de signaux asynchrones entre Main_Process et les Auxiliary_Processes.

Puisque plusieurs processus peuvent accéder aux variables en même temps, ces variables peuvent être verrouillées par LOCK et UNLOCK pour les lire et les modifier de manière cohérente (pour éviter l'affectation de nom simultanée d'un nœud sans nom depuis les deux directions, par exemple).

Les variables indexées (1,2) sont propres à chaque Auxiliary_Process.

Toutes les variables de la structure de données MyStatus sont considérées comme des variables partagées.

Toutes les autres variables partagées sont spécifiées dans le Tableau 11.

Tableau 11 – Variables partagées d'un nœud

Variable	Type	Signification
Detect (1) Detect (2)	BOOLEAN1 BOOLEAN1	‘1’ si la détection est activée dans Direction_1 ‘0’ si la détection est activée dans Direction_2
Found (1) Found (2)	BOOLEAN1 BOOLEAN1	mis à ‘1’: - par l’Auxiliary_Process qui a détecté un autre nœud, ou - par une Presence_Response dont RemStr n'est pas nul. remis à ‘0’: - lorsque le nœud est nommé par cette direction; - lorsque le nœud est mis en position intermédiaire, ou - lorsque le nœud est dénommé
Lost(1) Lost(2)	INTEGER8 INTEGER8	compteurs qui détectent la perte d'un nœud déjà détecté mais qui ne porte pas encore de nom
Insist	BOOLEAN1	‘1’ si le nœud d’extrémité insiste dans les deux directions, ‘0’ si une direction détecte une composition plus forte
LocStr	Composition_Strength	force de la composition à laquelle le nœud appartient
RemStr(1) RemStr(2)	Composition_Strength Composition_Strength	force d'une autre composition détectée par son propre Auxiliary_Process ou reçue dans le champ remote_strength d'une trame de supervision

5.5.4.4.4 Variables du Main_Process

Les variables figurant dans le Tableau 12 sont utilisées dans Main_Process. D'autres variables locales sont données dans les diagrammes SDL des macros ou des procédures correspondantes.

Tableau 12 – Variables du Main_Process

Variable	Type	Signification
Topo	RECORD	Topographie, avec une entrée par nœud nommé
TopoCounter	Topo_Counter	Voir 5.5.2.6
MasterTopo	Master_Topo	Voir 5.5.2.7
Inward	ANTIVALENT2	Direction du maître (pour les esclaves nommés)
Outward	ANTIVALENT2	Direction opposée au maître (pour les esclaves nommés)
NamingDir	ANTIVALENT2	Direction de nomination (0 = néant, 1 = descendant, 2 = ascendant) pour le maître.
Missed(1) Missed(2)	UNSIGNED8 UNSIGNED8	Détecte la perte d'un nœud d’extrémité dans chaque direction
Errors	UNSIGNED8	Compteur d’erreurs, augmenté par chaque situation défavorable et remis à zéro par une issue favorable de cette même situation, donc implicitement personnalisé pour chaque situation.
C_bit	BOOLEAN1	tant que ce bit est activé, toutes les Process_Data_Responses ou Message_Data_Responses activent leur C_bit.

Le fonctionnement du maître dépend des listes spécifiées dans le Tableau 13.

Tableau 13 – Listes du Main_Process

Variable	Type	Signification
Periodic_List [n]	RECORD	Liste des adresses à interroger pour les Données de Processus d'une Période de Base donnée, où n est la position de la Période de Base à l'intérieur de la Macro_Period.
Message_List	RECORD	Liste des nœuds à interroger pour les Données de Messagerie, classés dans l'ordre de détection des requêtes d'émission (signalé par le A_bit)
Supervisory_List	RECORD	Liste des nœuds à interroger pour les Données de Supervision, classés dans l'ordre de détection des requêtes d'émission (signalé par le C_bit)

5.5.4.5 Auxiliary_Process

5.5.4.5.1 Processus

Comme le montre la Figure 86, Auxiliary_Process détecte la présence de nœuds n'appartenant pas à la composition. Il est exécuté uniquement par les nœuds dans la position End_Setting. Les nœuds isolés disposent de deux Auxiliary_Process actifs. L'échange de trames a lieu sur le Canal Auxiliaire.

Auxiliary_Process se compose de deux états, un état DETECTING_RESPONSE auquel il n'accède que si le nœud est nommé et n'a pas cédé, et un état DETECTING_REQUESTS auquel il accède aussi longtemps que Auxiliary_Process est actif.

5.5.4.5.2 État DETECTING_RESPONSE

Un nœud nommé qui n'a pas cédé doit envoyer une Detect_Request et attendre une Detect_Response ou une Detect_Request à l'état DETECTING_RESPONSE.

Il doit attendre à l'état DETECTING_RESPONSE:

a) une temporisation T_detecting_response
   - et passer à l'état DETECTING_REQUESTS;

b) Detect_Response ou Detect_Request:
   - enregistrer la présence et la force de la composition éloignée (Found (this_dir)= '1'; Lost = 0);
   - comparer les forces respectives et céder éventuellement, si l'autre composition est plus forte, 'insist' = '0', et
   - passer à l'état DETECTING_REQUESTS;

c) autre type:
   - ignorer et passer à DETECTING_REQUESTS.

NOTE 1 Detect_Request est la seule Trame-Maître envoyée par un esclave.

NOTE 2 Certaines situations peuvent provoquer l'arrêt inconditionnel du nœud.

5.5.4.5.3 État DETECTING_REQUESTS

Auxiliary_Process doit attendre à l'état DETECTING_REQUESTS:

a) une temporisation T_detecting:

- si le nœud n'a pas reçu de trame pendant le temps T_detecting, il doit incrémenter le compteur ‘Lost’ de cette direction et passer à l'état DETECTING_RESPONSE;
- si le nœud n'a pas reçu de trame pendant les périodes de détection MAXLOST consécutives, il doit désactiver la variable ‘Found’, mettre ‘RemStr (this_dir)’ à zéro, puis passer à l'état DETECTING_RESPONSE.

b) Detect_Request:

- enregistrer la présence de l’autre nœud en mettant (Found(this_dir)= ‘1’, Lost = 0);
- comparer les forces respectives et, si la force du demandeur est plus importante ou équivalente, céder s’il n’est pas fort en mettant ‘Insist’ à ‘0’, et
- envoyer une Detect_Response;

c) Naming_Request:

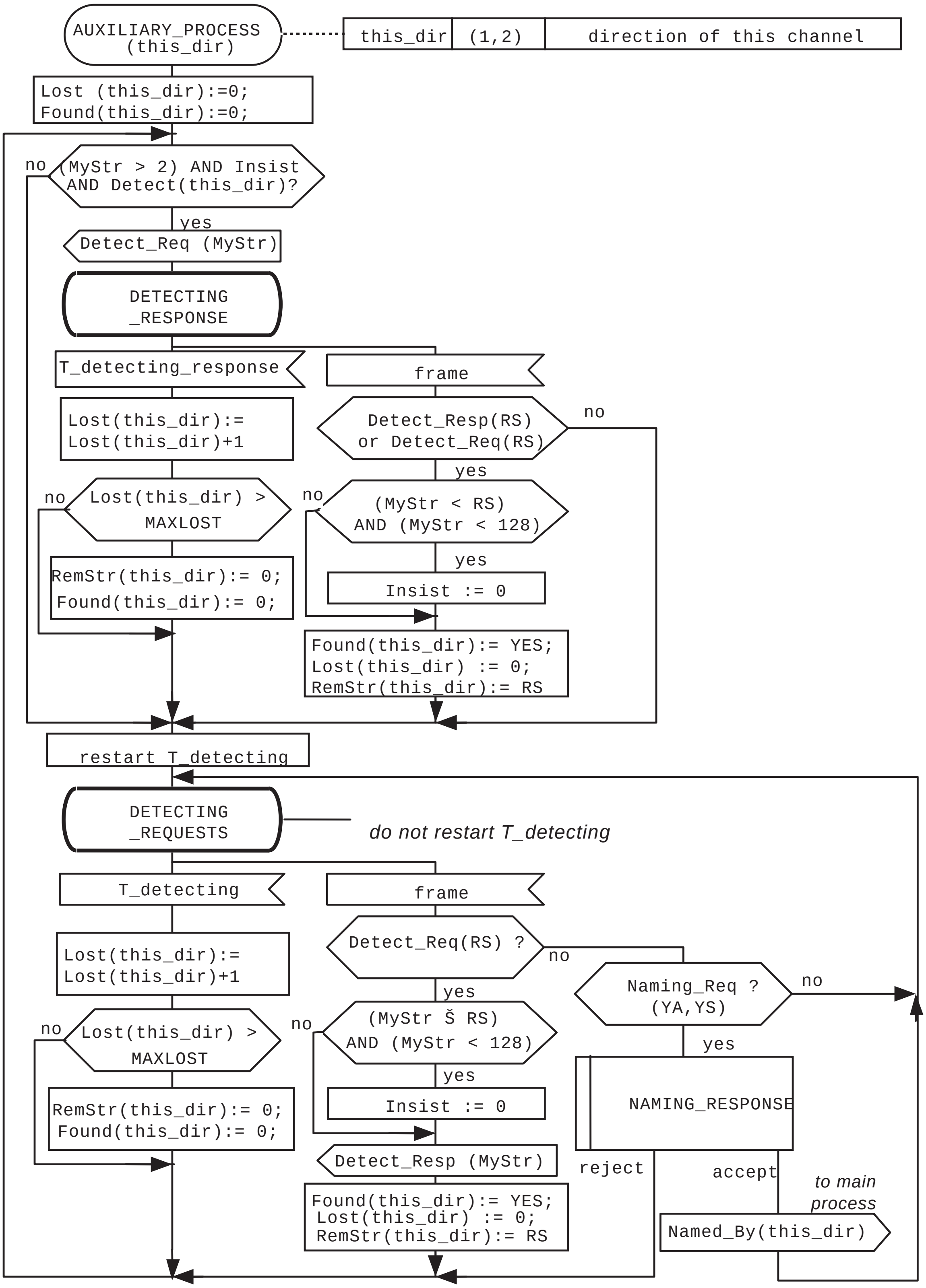
- si ‘Found(1)’, respectivement ‘Found(2)’ est ‘0’ et qu’il n’a donc pas reçu de Detect_Request d’une composition plus forte préalablement, il doit déclarer la ligne sur laquelle la trame a été reçue comme étant perturbée et se fier à l'autre ligne, même si elle est déjà perturbée;
- sinon, il doit exécuter la macro NAMING_RESPONSE qui, si elle aboutit, termine Auxiliary_Process;

d) autre type de trame:

- déclarer la ligne sur laquelle cette trame a été reçue comme étant perturbée et se fier à l'autre ligne, même si elle est déjà perturbée, puis passer à l’état DETECTING_REQUEST sans réinitialiser T_detecting.

NOTE 1 Detect_Request et Naming_Request sont les seuls types de trames qu'un nœud attend sur son Canal Auxiliaire. Si un nœud reçoit une Naming_Request sans avoir déjà reçu une Detect_Request, elle doit être considérée comme une erreur de protocole.

NOTE 2 Étant donné qu'une Detect_Request aurait pu être brouillée par une collision, le temps T_detecting est modulé de manière aléatoire autour d’une valeur médiane pour éviter les collisions répétitives.


AUXILIARY_PROCESS (this_dir)
this_dir
(1,2)
direction of this channel
Lost (this_dir):=0;
Found(this_dir):=0;
no
(MyStr > 2) AND Insist AND Detect(this_dir)?
yes
Detect_Req (MyStr)
DETECTING _RESPONSE
T_detecting_response
frame
Lost(this_dir):= Lost(this_dir)+1
Detect_Resp(RS) or Detect_Req(RS)
no
yes
no
Lost(this_dir) > MAXLOST
no
(MyStr < RS) AND (MyStr < 128)
yes
RemStr(this_dir):= 0;
Found(this_dir):= 0;
Insist := 0
Found(this_dir):= YES;
Lost(this_dir) := 0;
RemStr(this_dir):= RS
restart T_detecting
DETECTING _REQUESTS
do not restart T_detecting
T_detecting
frame
Lost(this_dir):= Lost(this_dir)+1
Detect_Req(RS) ?
no
yes
Naming_Req ? (YA,YS)
no
no
Lost(this_dir) > MAXLOST
no
(MyStr Š RS) AND (MyStr < 128)
yes
yes
RemStr(this_dir):= 0;
Found(this_dir):= 0;
Insist := 0
NAMING_RESPONSE
Detect_Resp (MyStr)
reject
accept
to main process
Found(this_dir):= YES;
Lost(this_dir) := 0;
RemStr(this_dir):= RS
Named_By(this_dir)

Légende

Anglais	Français
direction of this channel	direction de ce canal
yes	oui
frame	trame
Detect_Resp (RS) or Detect_Req (RS)	Detect_Resp (RS) ou Detect_Req (RS)
restart T_detecting	redémarrer T_detecting
do not restart T_detecting	ne pas redémarrer T_detecting
no	non
reject	rejeter
accept	accepter
to main process	vers le processus principal

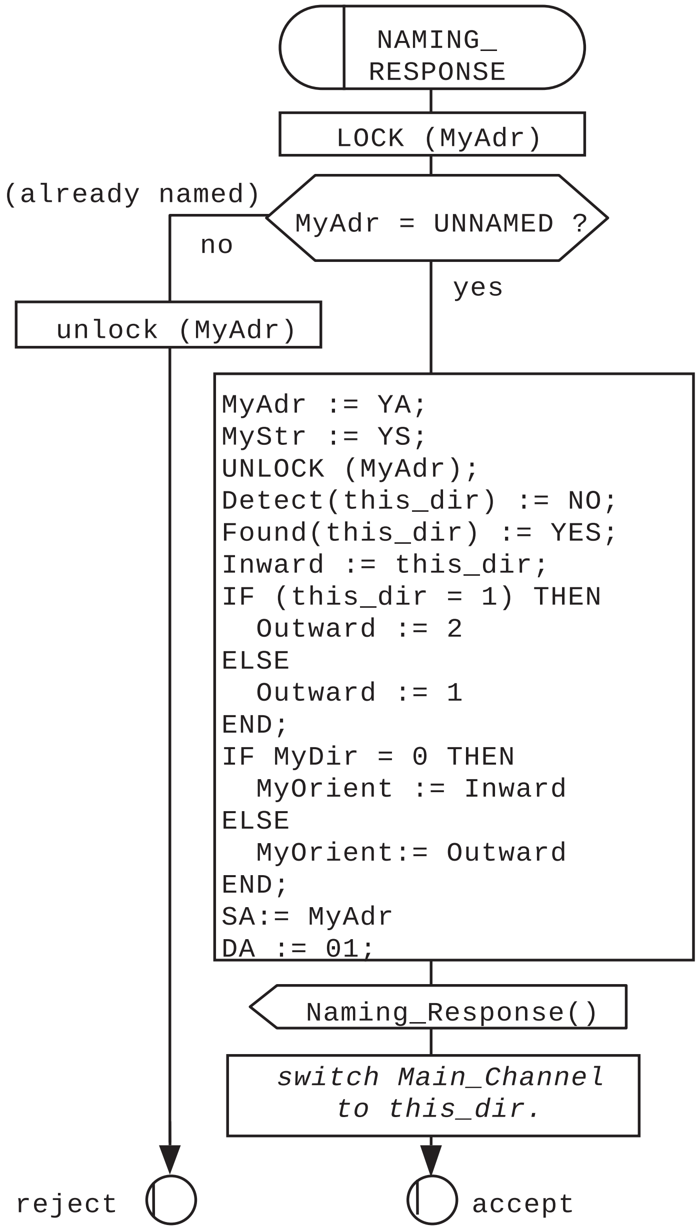
Figure 86 – Etats AUXILIARY_PROCESS

5.5.4.5.4 Macro NAMING_RESPONSE

La macro NAMING_RESPONSE, représentée dans la Figure 87, attribue une adresse à un nœud:

- si le nœud a déjà été nommé, il doit ignorer Naming_Request. Il convient que cette situation ne se produise que si le nœud a été nommé sur l'autre canal;
- si le nœud n'est pas nommé, il doit répondre par Naming_Response. Il doit arrêter la détection dans cette direction et attribuer le canal à son Canal Principal.

NOTE La vérification que le nœud n'est pas nommé demande une opération de lecture exclusive du nom MyAdr, puisque l'autre Auxiliary_Process peut également y accéder.



Légende

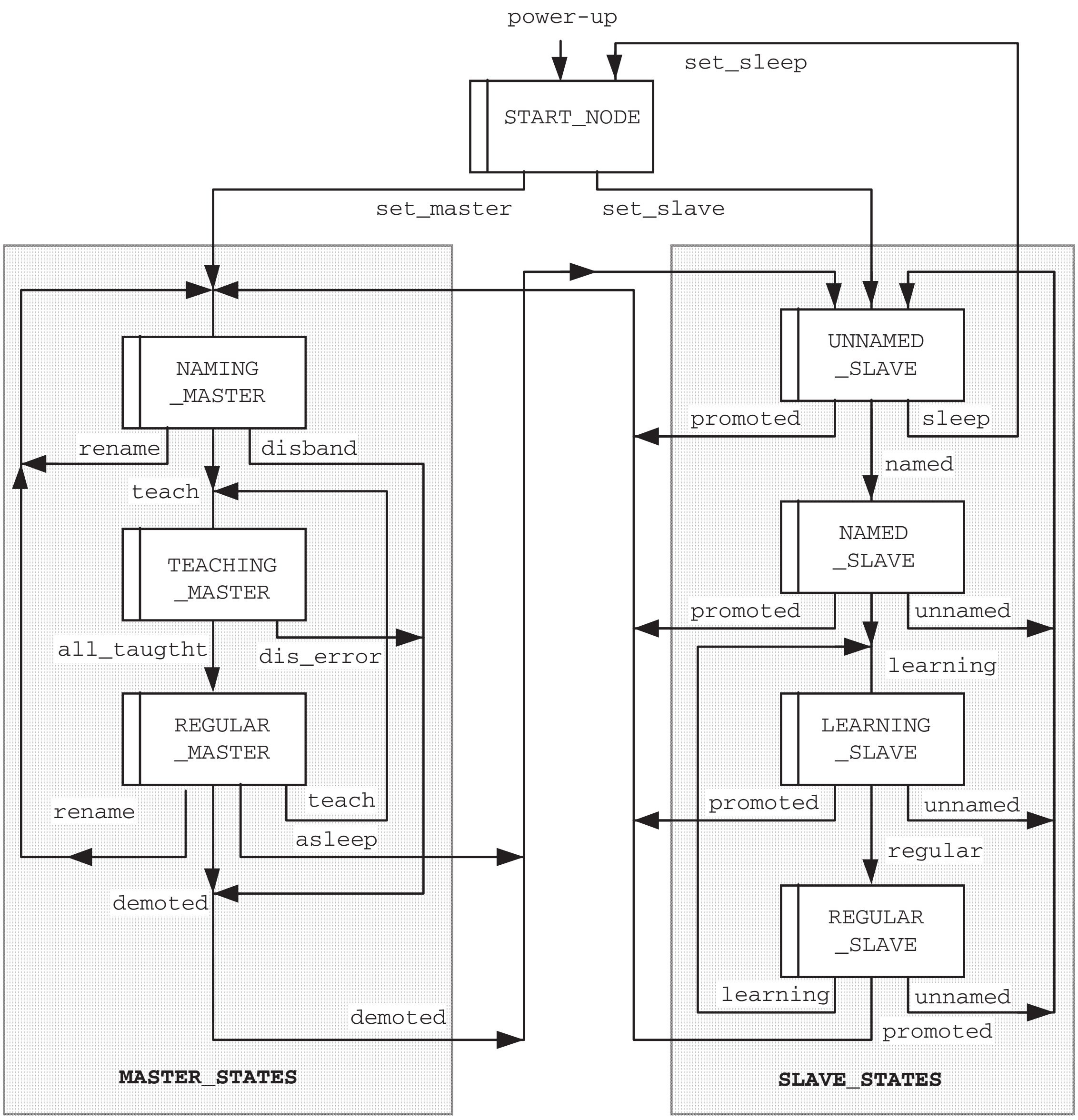
Anglais	Français
(already named)	(déjà nommé)
no	non
yes	oui
switch Main_Channel to this_dir	passer Main_Channel à this_dir
reject accept	rejeter accepter

Figure 87 – Macro NAMING_RESPONSE

5.5.4.6 Principaux états de Main_Process

Comme l'indique la Figure 88, Main_Process est divisé en états principaux.

Ces états sont eux-mêmes divisés en plusieurs états, représentés par une macro (qui n'apparaît qu'une seule fois) ou par une procédure (qui peut apparaître plusieurs fois), comme décrit dans les paragraphes suivants.



Légende

Anglais	Français
power-up	mise sous tension
rename	renommer
disband	dissoudre
teach	apprendre
asleep	en veille
demoted	dégradé
promoted	promu
sleep	veille
unnamed	sans nom
regular	régulier
learning	apprentissage

Figure 88 – Etats de MAIN_PROCESS

L'état START_NODE est décrit dans le Tableau 14.

Tableau 14 – START_NODE

START_NODE	Le nœud passe à l'état START_NODE suite à une défaillance, une remise à zéro ou une mise sous tension. Il démarre dans un état hors tension dans lequel le nœud est passif ou depuis un état à basse puissance, dans lequel le nœud attend un signal du bus ou un signal de l'application pour restaurer la pleine puissance. Le nœud est ensuite configuré par l'application grâce à la commande NodeSetUp. Si le nœud a été configuré comme nœud fort, il passe immédiatement à l'état NAMING_MASTER. S'il est configuré comme nœud faible ou esclave, il passe à l'état UNNAMED_SLAVE

Le Tableau 15 donne les états dans lesquels un nœud fonctionne en tant que maître.

Tableau 15 – MASTER STATES (État de Maître)

NAMING_MASTER	Initialise le nœud comme maître isolé. Met ou laisse le nœud dans la position End_Setting avec les deux canaux de détection actifs. Envoie des trames de détection dans les deux directions à la recherche d'autres nœuds. Lorsque le maître détecte un nœud, il le nomme et met à jour sa liste de Node_Status. Quand il ne trouve plus de nœuds à nommer dans les deux directions, le maître arrête la nomination et passe à l'état TEACHING_MASTER. Si un maître faible trouve une composition plus forte, il est dégradé et repasse à l'état UNNAMED_SLAVE. Il doit en premier lieu dénommer les nœuds qu'il a nommés (dissolution). Si le maître rencontre une erreur irrécupérable, il relance la procédure en tant que maître isolé. Il n'y a pas de limite de temps pour cet état, puisqu'il peut s'agir d'une situation normale (rame seule).
TEACHING_MASTER	Le maître distribue la Topographie, en demandant un par un à chaque nœud nommé de diffuser son descripteur et ses données d'inauguration à tous les autres noeuds. Si cette diffusion réussit, le maître passe à l'état REGULAR_MASTER. Si la diffusion ne réussit pas, il revient à l'état NAMING_MASTER.
REGULAR_MASTER	Le maître interroge les nœuds en fonctionnement normal. Le maître traite les situations exceptionnelles suivantes: - changement de descripteur ou d'état d'un nœud; - changement de force du maître. Le maître sort de l'état REGULAR_MASTER en cas de: - raccourcissement du bus (le Nœud d'Extrémité n'est plus détecté); - prolongement du bus (nomination de nœuds supplémentaires)

Le Tableau 16 répertorie les états dans lesquels le nœud fonctionne comme esclave (états SLAVE).

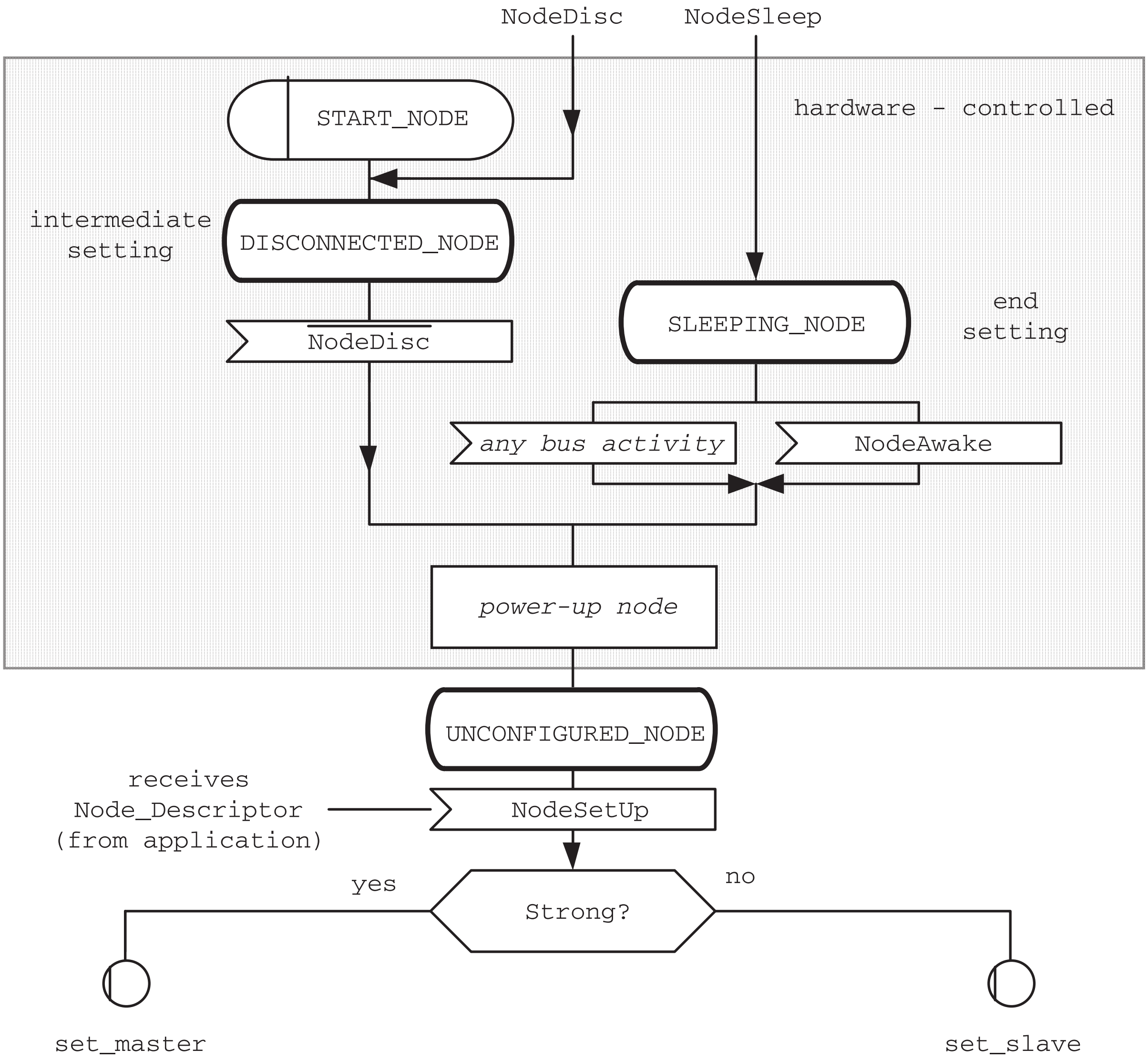
Tableau 16 – SLAVE STATES (états esclave)

UNNAMED_SLAVE	Un UNNAMED_SLAVE est en position End_Setting, ses deux Auxiliary_Processes étant à l'écoute (sans émettre). Il quitte cet état lors de la réception d'une trame de nomination depuis un maître ou après avoir été promu au rang de nœud fort ou faible par l'application.
NAMED_SLAVE	Naming_Request conduit le nœud à commuter son Canal Principal dans la direction du maître. Son Auxiliary_Process dans l'autre direction reste actif. Le maître envoie régulièrement une Status_Request pour chercher d'autres nœuds à son extrémité ouverte. Lors de la détection d'un nœud supplémentaire sans nom, le maître bascule le nœud sur la position Intermediate_Setting, ce qui désactive l'Auxiliary_Process. Le nœud quitte cet état quand il reçoit une Topography_Request ou une Topograhy_Response, qui signale la fin de la phase de nomination.
LEARNING_SLAVE	L'esclave nommé reçoit les données d'inauguration de tous les autres nœuds et diffuse ses propres données d'inauguration à tous les autres nœuds. Il augmente son compteur d'inauguration de 1. Le nœud quitte cet état en recevant une Presence_Request, par laquelle le maître signale qu'il passe en fonctionnement normal, ou toute trame indiquant un fonctionnement normal
REGULAR_SLAVE	Dans cet état, le nœud est supposé avoir été mis à jour avec la Topographie complète. S'il s'agit d'un Nœud d'Extrémité, son Auxiliary_Process actif contrôle le prolongement du bus, qu'il signale au maître par sa Presence_Response. Tous les nœuds contrôlent la présence des deux Nœuds d'Extrémité. Un nœud retourne à l'état UNNAMED_SLAVE s'il ne détecte plus les Nœuds d'Extrémité pendant trois Presence_Responses consécutives. S'il a été mis à jour, le nœud s'attend à être interrogé régulièrement pour ses Données de Processus et à recevoir le Process_Data des autres nœuds. Il retourne à l'état LEARNING_SLAVE s'il reçoit une Topography_Response ou une Topography_Request adressée à lui-même, par laquelle le maître signale un changement de composition sans changement d'adresse.

Dans tous les états esclave, un esclave peut être promu au rang de maître fort et passer directement à l'état NAMING_MASTER.

5.5.4.7 Macro START_NODE

Comme le montre la Figure 89, cette macro définit les états matériels et logiciels par lesquels un nœud passe avant de devenir maître ou esclave.



Légende

Anglais	Français
hardware - controlled	contrôlé par le matériel
intermediate setting	état intermédiaire
end setting	état d'extrémité
any bus activity	activité d'un bus
power-up node	nœud de mise sous tension
receives Node_Descriptor (from application)	reçoit Node_Descriptor (de l'application)
yes	oui
no	non
Strong?	Fort ?

Figure 89 – Macro START_NODE

5.5.4.7.1 État DISCONNECTED_NODE

Un nœud doit passer à cet état sans condition lorsque l'application active le signal NodeDisc, indiquant que le nœud va être mis hors tension ou qu'il a subi une perturbation grave.

Il s'agit de l'état initial d'un nœud hors tension. Un nœud à l'état DISCONNECTED_NODE doit être dans la position Intermediate_Setting.

Un nœud doit quitter cet état lorsque l'application le met sous tension et désactive le signal NodeDisc. Ensuite, il doit passer à l'état UNCONFIGURED_NODE.

5.5.4.7.2 État SLEEPING_NODE

Un nœud doit passer à cet état lorsque la commande NodeSleep est activée et le signal NodeDisc est désactivé.

Un nœud doit être dans la position End_Setting, dans un état de basse puissance.

Un nœud doit quitter cet état:

a) lorsque l'application active la commande NodeAwake, ou

b) lorsqu'il détecte une activité sur le bus.

Lorsqu'un nœud quitte l'état de veille, il doit activer le signal MySwitchOn pour mettre le nœud sous tension et passer à l'état UNCONFIGURED_NODE. Le signal MySwitchOn doit être activé par le matériel pour tous les états, à l'exception de DISCONNECTED_NODE et SLEEPING_NODE.

NOTE 1 Le passage à cet état perturbe le bus. Le nœud serait de nouveau réveillé en cas d'activité sur le bus. Il convient donc que l'application maintienne le signal NodeSleep suffisamment longtemps pour éviter qu'un autre nœud ne réveille le nœud.

NOTE 2 Le commutateur de bus est sous tension, car une perte complète d'énergie amènerait le nœud en Intermediate_Setting.

NOTE 3 'Toute activité sur le bus' est définie par un Carrier_Sense sans signal SQE (voir 4.7.1.5) ou tout autre signal qui indique qu'une trame bien formée a été reçue sur l'un des canaux.

5.5.4.7.3 État UNCONFIGURED_NODE

Un nœud doit être en position Intermediate_Setting et attendre son Node_Descriptor et ses donnés d'inauguration.

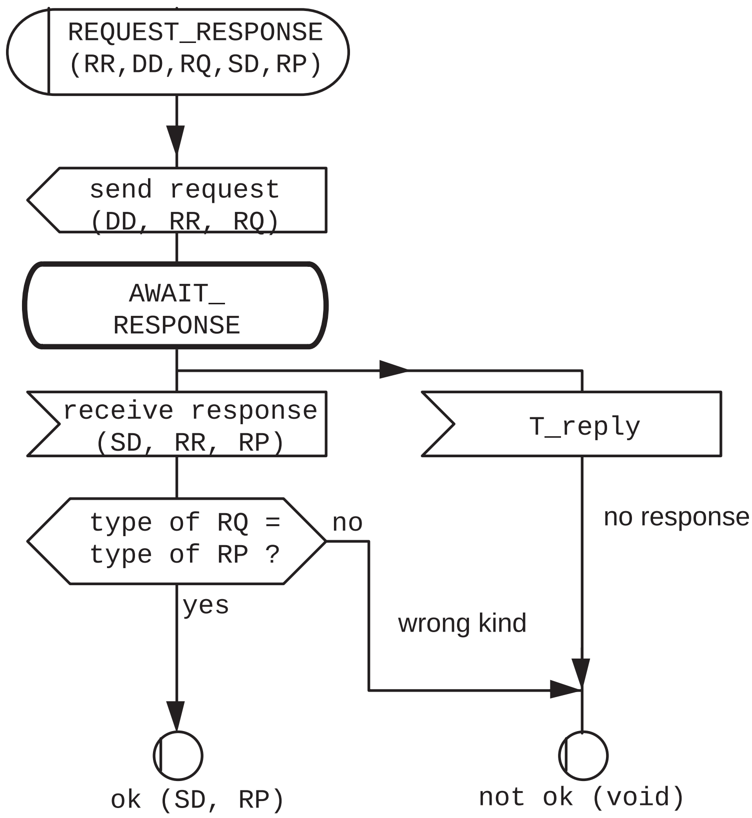
Un nœud doit quitter l'état UNCONFIGURED_NODE à la réception d'une commande NodeSetUp avec un Node_Descriptor valide.

Il doit ensuite passer à la macro NAMING_MASTER s'il s'agit d'un nœud fort ou à la macro UNNAMED_SLAVE s'il s'agit d'un nœud faible ou d'un nœud esclave.

5.5.4.8 États Maîtres

5.5.4.8.1 Procédure REQUEST_RESPONSE

Comme l'indique la Figure 90, cette procédure envoie une demande et attend la réponse correspondante.



Légende

Anglais	Français
send request	envoyer la demande
receive response	recevoir la réponse
type of RQ =	type de RQ =
type of RP ?	type de RP ?
no	non
yes	oui
no response	pas de réponse
wrong kind	mauvais type
not ok (void)	pas ok (vide)

Figure 90 – Procédure REQUEST_RESPONSE

Les paramètres sont les suivants:

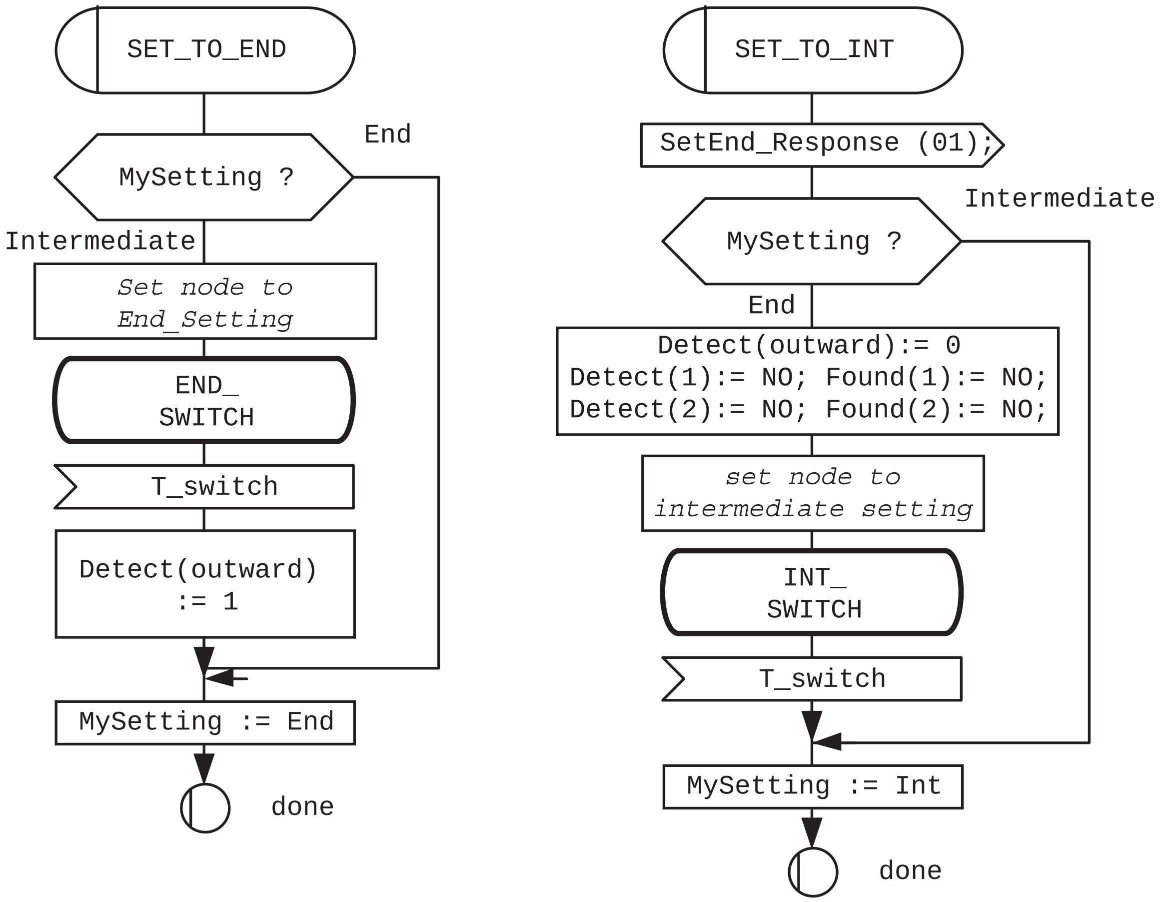
a) RR: Type de Requête/Réponse, une parmi: {Presence, Status, Naming, Topography};
b) DD: Destination_Device;
c) SD: Source_Device;
d) RP: Paramètres de réponse (dépend de RR, selon la définition de la trame);
e) RQ: Paramètres de requête (dépend de RR, selon la définition de la trame).

Cette procédure se termine:

- à la réponse de l'esclave, ou
- à l’échéance de la temporisation T_reply.

5.5.4.8.2 Procédures SET_TO_END et SET_TO_INT

Ces deux procédures mettent le nœud à l'état de End_Setting ou Intermediate_Setting, respectivement, s'il n'est pas déjà dans cet état (voir la Figure 91).



Légende

Anglais	Français
End	Extrémité
Intermediate	Intermédiaire
Set node to End_Setting	mettre le nœud en état d'extrémité
set node to intermediate setting	mettre le nœud en état intermédiaire
done	terminé

Figure 91 – Procédures SET_TO_INT et SET_TO_END

Les variables Detect(1) ou Detect(2) commandent respectivement le démarrage et l'arrêt de l'AUXILIARY_PROCESS.

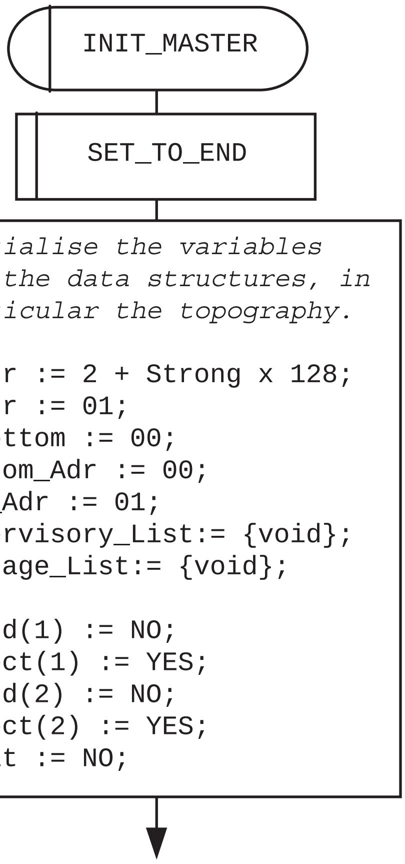
Si le nœud à mettre en End_Setting est le maître lui-même ou si le nœud ne porte pas de nom, Detect(1) et Detect(2) prennent la valeur 1.

5.5.4.8.3 Macro INIT_MASTER

Comme le montre la Figure 92, cette macro configure un nœud comme maître, suite à sa promotion d'esclave à maître fort ou au manque d'activité sur le bus qui laisse un esclave sans nom devenir maître.

Le nœud s'initialise d'abord en position End_Setting (si ce n'est pas déjà le cas), configure son adresse et sa force, installe une Topographie vide et active ses deux canaux de détection, qui commencent à envoyer des Detect_Requests. Dans cette configuration, le maître se comporte comme s'il s'interrogeait lui-même.

Après cette phase, le maître est prêt à nommer d'autres nœuds qu'il pourrait trouver.



LégendeLégende

Anglais	Français
initialise the variables and the data structures, in particular the topography	initialise les variables et les structures de données, en particulier la topographie

Figure 92 – Macro INIT_MASTER

5.5.4.8.4 Macro NAMING_MASTER

Comme le montre la Figure 93, cette macro contient les états dans lesquels le maître nomme les autres nœuds. Le maître entre dans cet état en exécutant la macro INIT_MASTER, qui l'initialise.

Le maître doit attendre dans l'état AWAIT_PERIOD le début d'une période T_naming_master. Si cette période est déjà écoulée, le maître continue immédiatement.

Le maître doit exécuter la macro ASK_END, qui demande à un Nœud d'Extrémité s'il a découvert une autre composition, et agit selon le résultat de cette macro:

a) si la composition détectée est plus forte, le maître doit dissoudre sa composition en allant à la macro UNNAMING_MASTER;

b) si la composition détectée est plus faible ou de force égale, le maître doit attendre la dissolution de l'autre composition (ce qui prend un temps non défini);

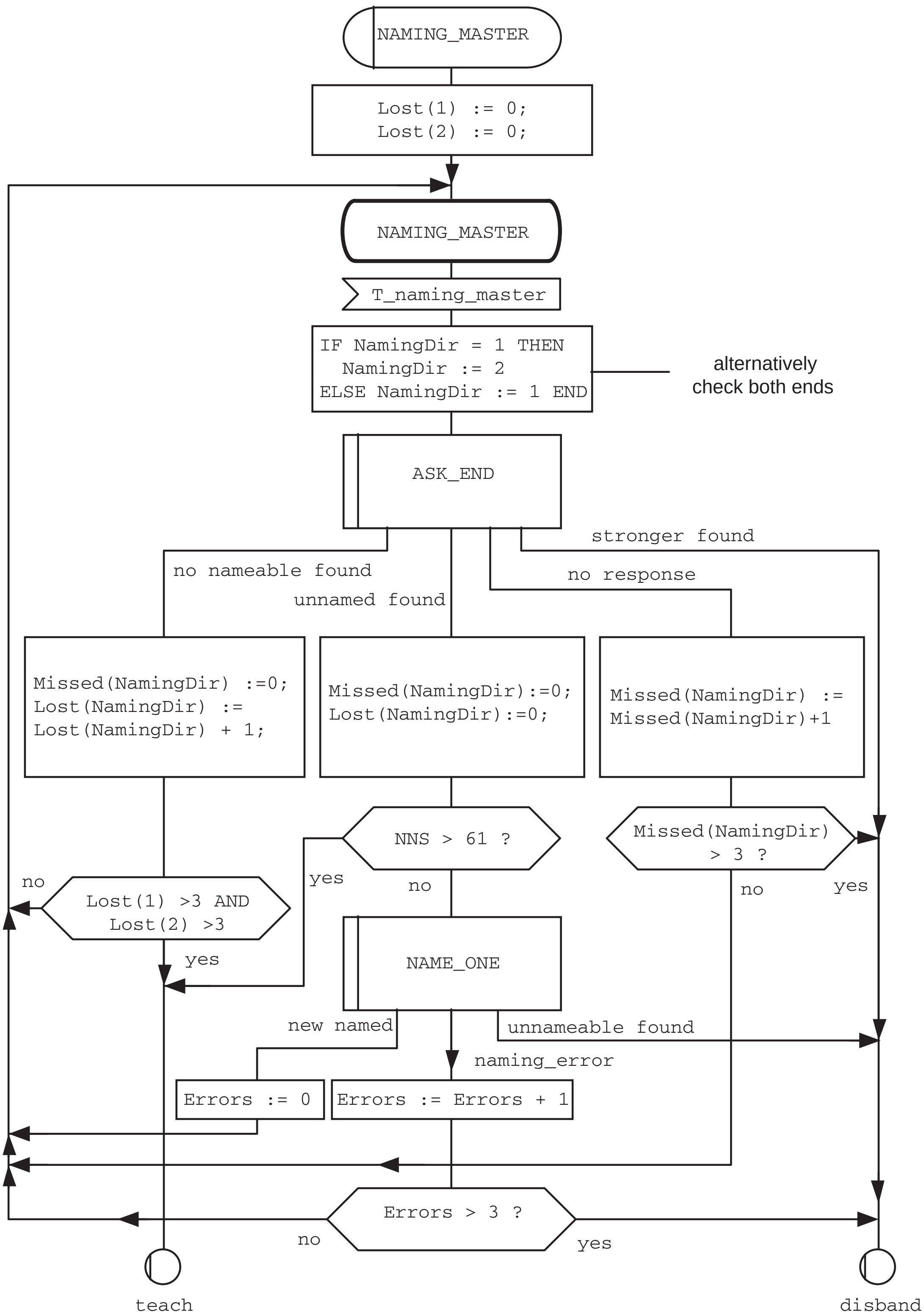
c) si le nœud d’extrémité ne répond pas, le maître doit enregistrer une erreur et faire une nouvelle tentative ultérieurement. Trois erreurs consécutives dans la même direction doivent provoquer la dissolution de sa composition par l'exécution de la macro UNNAMING_MASTER;

d) si le nœud d’extrémité renvoie un nœud sans nom, le maître doit le nommer comme nouveau nœud d’extrémité;

e) s’il ne détecte pas de nœud à nommer pendant trois interrogations consécutives dans la même direction, il doit passer à l’état TEACHING_MASTER.

NOTE 1 Le maître nomme un seul nœud par période T_naming_master.

NOTE 2 Dans l’état NAMING_MASTER, l’inauguration est toujours permise.


NAMING_MASTER
Lost(1) := 0;
Lost(2) := 0;
NAMING_MASTER
T_naming_master
IF NamingDir = 1 THEN
NamingDir := 2
ELSE NamingDir := 1 END
alternatively
check both ends
ASK_END
stronger found
no nameable found
unnamed found
no response
Missed(NamingDir) :=0;
Lost(NamingDir) :=
Lost(NamingDir) + 1;
Missed(NamingDir):=0;
Lost(NamingDir):=0;
Missed(NamingDir) :=
Missed(NamingDir)+1
NNS > 61 ?
Missed(NamingDir)
> 3 ?
no
yes
no
no
yes
Lost(1) >3 AND
Lost(2) >3
yes
NAME_ONE
new named
unnameable found
naming_error
Errors := 0
Errors := Errors + 1
Errors > 3 ?
no
yes
teach
disband

Légende

Anglais	Français
alternatively check both ends	vérifier les deux extrémités en alternance
no nameable found	aucun élément à nommer trouvé
unnamed found	élément sans nom trouvé
stronger found	élément plus fort trouvé
no response	pas de réponse
no	non
yes	oui
new named	élément nouvellement nommé
teach	apprentissage
disband	dissoudre
Errors > 3 ?	Erreurs > 3 ?

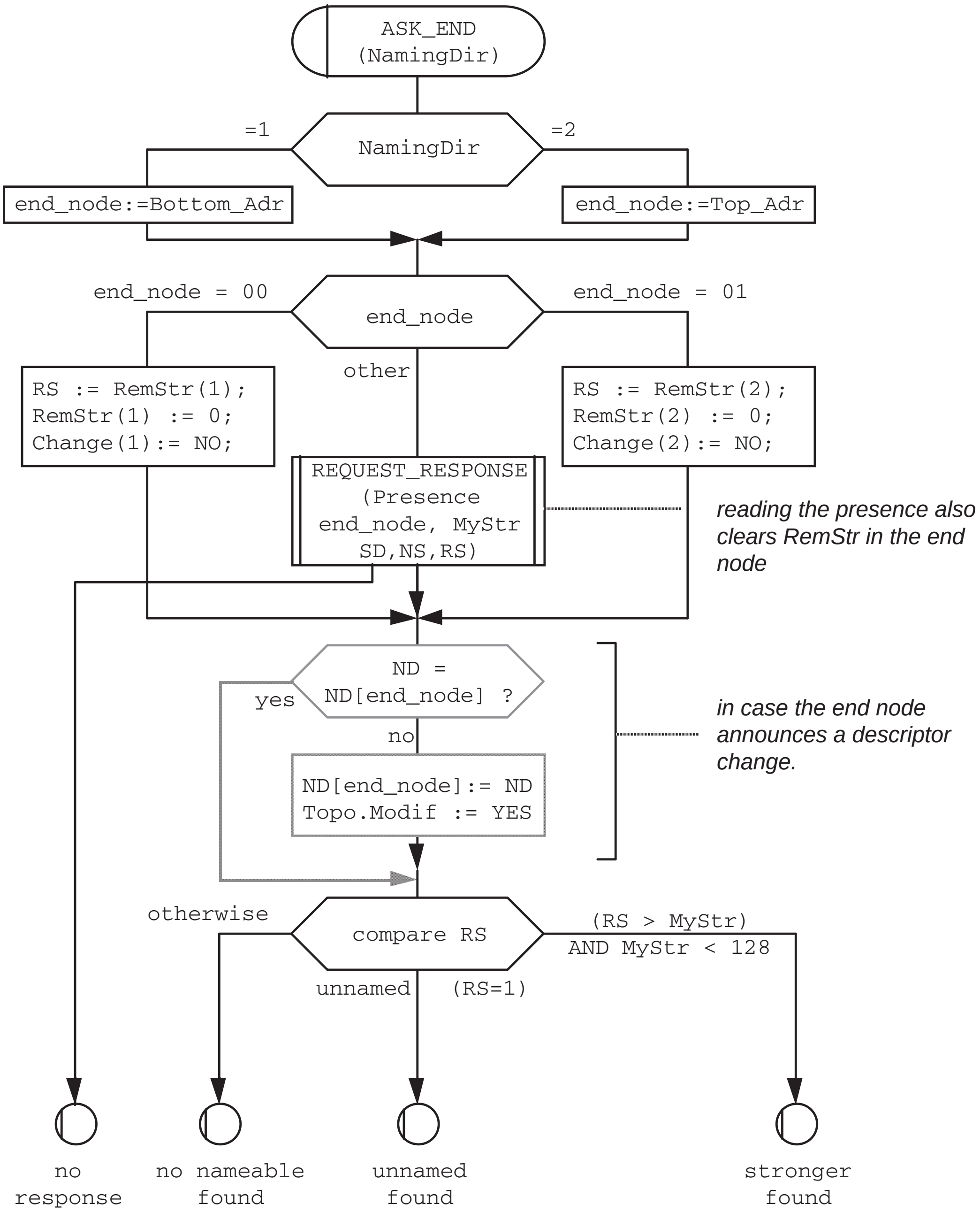
Figure 93 – Macro NAMING_MASTER

5.5.4.8.5 Macro ASK_END

Comme le montre la Figure 94, cette macro vérifie la présence du Nœud d'Extrémité dans la direction NamingDir et lui demande de signaler une éventuelle composition éloignée. Cette macro comprend implicitement le cas où le maître est lui-même un Nœud d'Extrémité.

Selon la force éloignée détectée, le maître distingue trois cas:

a) aucun nœud à nommer trouvé (aucun nœud ou un nœud déjà nommé);
b) nœud sans nom trouvé (le nœud éloigné accepte la nomination);
c) nœud le plus fort trouvé (le nœud éloigné appartient à une composition plus forte).



ASK_END (NamingDir)
NamingDir
=1
=2
end_node:=Bottom_Adr
end_node:=Top_Adr
end_node
end_node = 00
end_node = 01
other
RS := RemStr(1); RemStr(1) := 0; Change(1):= NO;
RS := RemStr(2); RemStr(2) := 0; Change(2):= NO;
REQUEST_RESPONSE (Presence end_node, MyStr SD,NS,RS)
reading the presence also clears RemStr in the end node
ND = ND[end_node] ?
yes
no
ND[end_node]:= ND Topo.Modif := YES
in case the end node announces a descriptor change.
compare RS
otherwise
unnamed
(RS=1)
(RS > MyStr) AND MyStr < 128
no response
no nameable found
unnamed found
stronger found

Légende

Anglais	Français
other	autre
reading the presence also clears RemStr in the end node	la détection de la présence efface également RemStr du nœud d'extrémité
in case the end node announces a descriptor change	si le nœud d'extrémité annonce une modification du descripteur
yes	oui
no	non
otherwise	sinon
no response	pas de réponse
no nameable found	aucun élément à nommer trouvé
unnamed found	élément sans nom trouvé
unnamed	sans nom
stronger found	élément plus fort trouvé

Figure 94 – Macro ASK_END

5.5.4.8.6 Procédure NAME_ONE

Comme le montre la Figure 95, cette procédure de la macro NAMING_MASTER est appelée lorsque le Nœud d'Extrémité a signalé la présence d'un nœud sans nom.

Cette procédure comporte un paramètre, la direction de nomination.

Si le maître est lui-même le Nœud d'Extrémité et qu'il a déjà nommé des nœuds de l'autre côté, il doit passer à Intermediate_Setting sans envoyer de commande sur le bus.

Sinon, le maître doit rester à End_Setting et envoyer une SetInt_Request au Nœud d'Extrémité dans la direction de nomination.

Si le maître ne reçoit pas de SetInt_Response, il doit attendre T_switch et répéter SetInt_Request.

Si le maître ne reçoit pas de SetInt_Response après trois tentatives, il doit quitter cette procédure avec 'naming_error'.

Sinon, après avoir reçu SetInt_Response, le maître doit attendre un délai T_switch, puis envoyer au nœud sans nom une Naming_Request comprenant l'adresse du nœud sous la forme 'your_address' et sa Force de Composition sous la forme 'your_strength', selon le protocole suivant:

a) un maître doit envoyer une Naming_Request avec Destination_Device = 'sans nom' pour la première tentative;
b) si le maître ne reçoit pas de Naming_Response à cette première tentative, il doit attendre un temps T_aux_main et renvoyer une Naming_Request avec comme Destination_Device l'adresse de nœud attribuée;
c) si le maître ne reçoit pas de Naming_Response à cette deuxième tentative, il doit renvoyer une Naming_Request avec comme Destination_Device l'adresse 'sans nom';
d) si le maître ne reçoit pas de Naming_Response à cette troisième tentative, il doit attendre un temps T_aux_main et renvoyer une Naming_Request avec comme Destination_Device l'adresse de nœud attribuée;

e) si le maître ne reçoit pas de Naming_Response à la quatrième tentative dans la limite T_reply, il doit renvoyer une Naming_Request avec comme Destination_Device l'adresse ‘sans nom’;

f) si le maître ne reçoit pas de Naming_Response à cette cinquième tentative, il doit attendre un temps T_aux_main et renvoyer une Naming_Request avec comme Destination_Device l'adresse de nœud attribuée;

g) si le maître ne reçoit pas de réponse dans la limite T_reply à cette sixième tentative, il doit restaurer l'ancien Nœud d’Extrémité à End_Setting en envoyant une SetEnd_Request et attendre T_switch pour la réouverture du commutateur avant de quitter la procédure NAME_ONE avec l'état ‘unnameable_found’.

Sinon, si la nomination réussit, le maître doit mettre à jour la topographie, les adresses des Nœuds d’Extrémité et la force de la composition, puis attendre T_aux_main et envoyer une Status_Request au nœud qui vient d’être nommé.

Si le maître reçoit la Status_Response sur la Trusted_Line uniquement (l’Observed_Line étant perturbée), il doit attendre 1,2 ms avant de répéter la Status_Request (et attendre la Status_Response), répétant cette procédure trois fois pour déterminer la qualité de la ligne et communiquer le résultat dans son prochain Master_Report.

S'il ne reçoit de réponse à aucune de ces trois tentatives d’envoi de Status_Request, il doit quitter la procédure avec le statut ‘unnameable_found’.

Sinon, s’il reçoit une Status_Response, il doit mettre à jour la Topographie, les adresses des Nœuds d’Extrémité et la force de la composition, puis quitter la procédure avec le statut ‘new_named’.

NOTE 1 Un nœud sans nom reçoit normalement une Naming_Request et envoie sa Naming_Response sur le Canal Auxiliaire.

NOTE 2 Le délai T_aux_main permet de commuter du Canal Auxiliaire vers le Canal Principal. Pendant la commutation, le trafic sur le bus est interrompu.

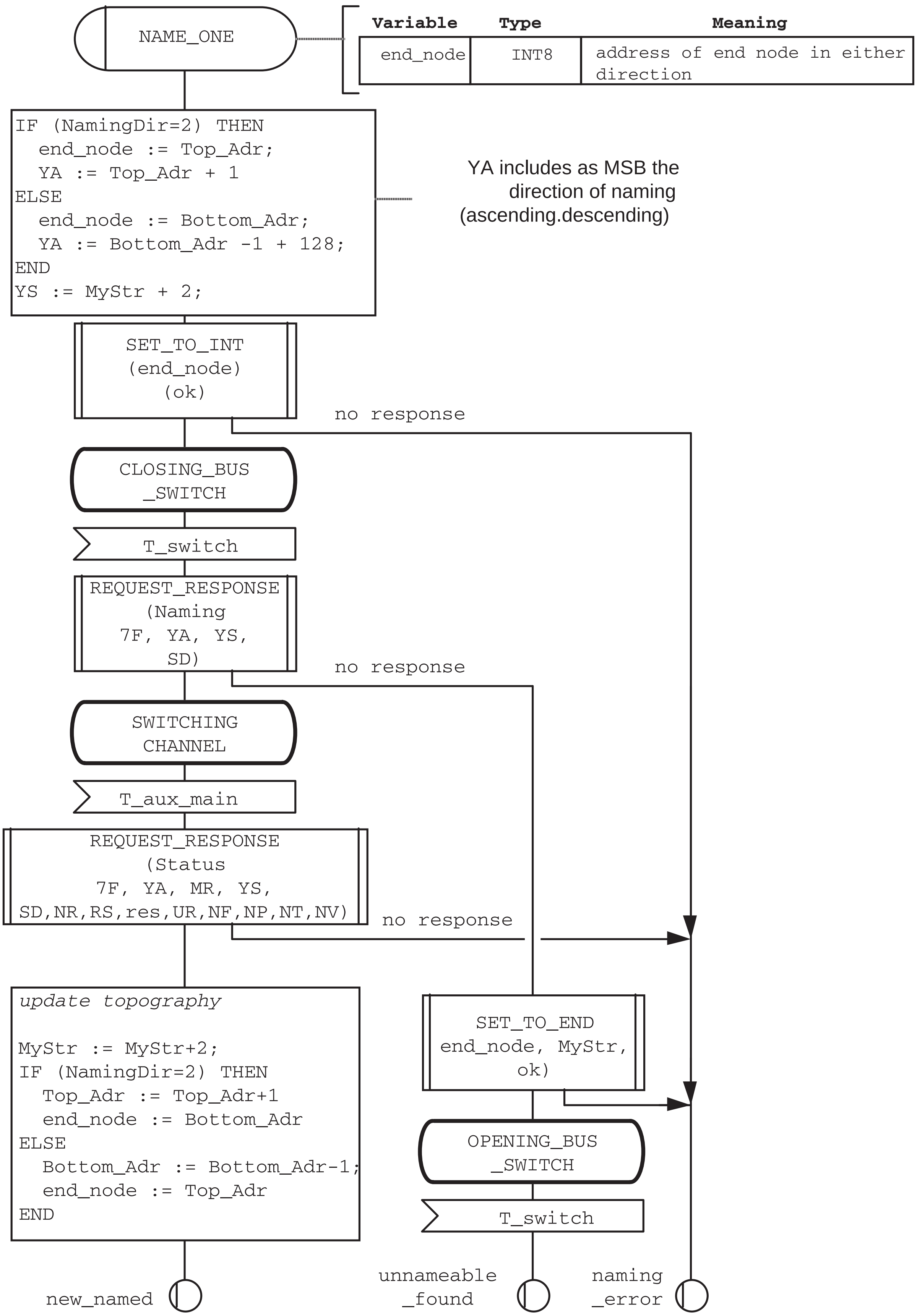
NOTE 3 Le nœud qui vient d'être nommé accepte la nomination par une Naming_Response ou la refuse en ne répondant pas.

NOTE 4 Si le nœud a été nommé mais que la réponse a été perdue, les tentatives avec Destination_Device = ‘sans nom’ ne peuvent aboutir. A l’inverse, si l'affectation d'un nom n'a pas eu lieu, les tentatives avec Destination_Device représentant l'adresse de nœud attribuée ne peuvent réussir.

NOTE 5 Le nœud nommé renvoie la Status_Response avec son Node_Descriptor et la force de la composition éloignée d’autres nœuds qu'il a pu détecter.

NOTE 6 Les trois répétitions de Status_Request en cas de perturbation permettent au maître de distinguer entre une perte de trame et une perte de ligne redondante entre lui-même et le Nœud d’Extrémité.

Variable	Type	Meaning
end_node	INT8	address of end node in either direction



Légende

Anglais	Français
Meaning	Signification
address of end node in either direction	adresse du nœud d'extrémité dans l'une ou l'autre direction
YA includes as MSB the direction of naming (ascending.descending)	YA intègre comme MSB la direction de la dénomination (ascendant.descendant)
no response	pas de réponse
SWITCHING CHANNEL	CANAL DE BASCULEMENT

Figure 95 – Procédure NAME_ONE

5.5.4.8.7 Macro TEACHING_MASTER

Comme le montre la Figure 96, le maître distribue les informations de la Topographie à tous les nœuds nommés lorsqu'il ne détecte plus de nœuds à nommer ou qu'il détecte un nœud qui signale un changement de son Node_Descriptor.

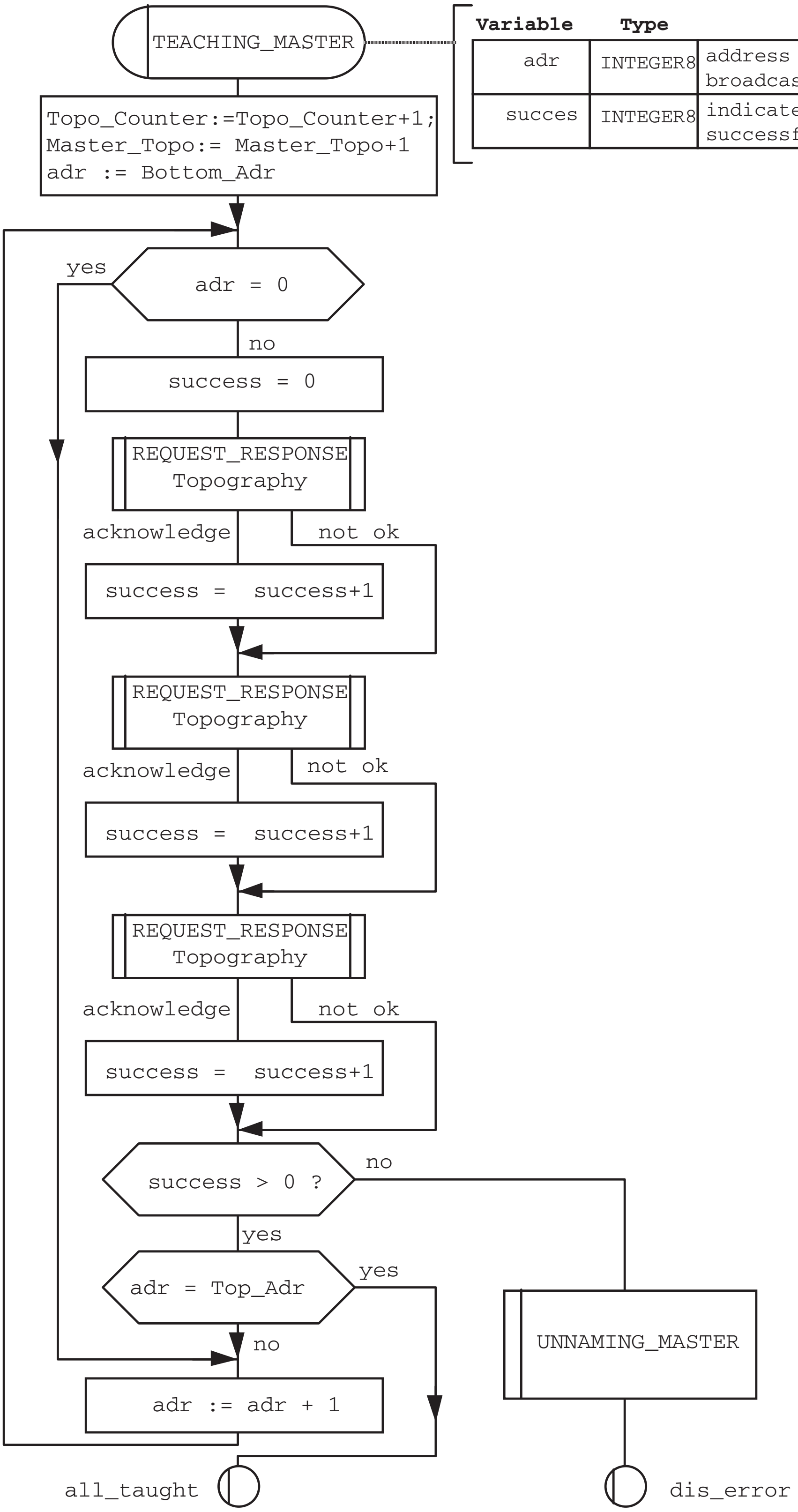
A cet effet, le maître doit incrémenter Topo_Counter et Master_Topo.

Le nœud doit envoyer trois fois Topography_Request à tous les nœuds nommés en séquence, en commençant par le nœud du bas et en terminant par le nœud du haut (y compris lui-même = auto-interrogation), dans l'ordre croissant des adresses.

Le nœud doit quitter cet état:

- s'il ne reçoit pas de Topography_Response après trois envois de Topography_Request, et il doit alors passer à la macro UNNAMING_MASTER;
- lorsque tous les nœuds ont envoyé leur Topography_Response, il doit alors passer à la macro REGULAR_MASTER.

Variable	Type	Meaning
adr	INTEGER8	address of the node asked to broadcast its Topography_Respon
succes	INTEGER8	indicates the number of successful distributions



Légende

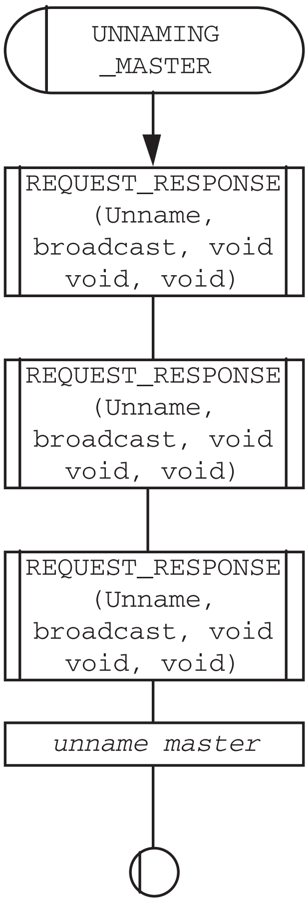
Anglais	Français
Meaning	Signification
address of the node asked to broadcast its Topography_Response	adresse du nœud chargé de la diffusion de sa Topography_Response
indicates the number of successful distributions	indique le nombre de distributions réussies
yes	oui
no	non
acknowledge	accusé de réception
not ok	pas ok
success > 0 ?	succès > 0 ?

Figure 96 – Macro TEACHING_MASTER

5.5.4.8.8 Macro UNNAMING_MASTER

Comme le montre la Figure 97, le maître dissout sa composition en dénommant tous les nœuds. Pour cela, il doit diffuser trois Unname_Requests de suite en moins de 1,0 ms puis, s'il est un nœud fort, passer à la macro NAMING_MASTER ou sinon, à la macro UNNAMED_SLAVE.

NOTE Le délai est nécessaire pour permettre à tous les nœuds de recevoir Unname_Request, un esclave observera le même délai avant de commuter en position End_Setting.



Légende

Anglais	Français
unname master	maître sans nom

Figure 97 – Macro UNNAMING_MASTER

5.5.4.8.9 Macro REGULAR_MASTER

Comme le montre la Figure 98, le maître est en fonctionnement normal.

La macro REGULAR_MASTER est divisée en trois blocs:

a) ‘PERIODIC_POLL’: le maître interroge les nœuds dans sa Periodic_List pour les Données de Processus. Cette phase est décrite en 5.5.4.8.11;

b) ‘SUPERVISORY_POLL’:

- dans chaque Période de base, le maître doit envoyer une Presence_Request à un Nœud d’Extrémité. Le maître doit maintenir un intervalle de moins de 6,5 . T_bp entre quatre envois successifs de Presence_Request dans la même direction. Un moyen pratique consiste à envoyer Presence_Request à un pourcentage fixe de la Période de Base (au début, par exemple),
- le maître doit demander l’état des nœuds qui ont activé le bit ‘C’. Ce contrôle doit avoir lieu après la Phase Périodique;

c) ‘MESSAGE_POLL’: le maître interroge les nœuds pour les Données de Messagerie pendant le temps qui reste avant le début de la Phase Périodique suivante (voir 5.5.4.8.12).

Le maître doit quitter la macro ‘REGULAR_MASTER’:

d) s'il ne détecte plus la présence d'un Nœud d’Extrémité pendant trois interrogations consécutives, puis dissoudre sa composition et passer à la macro ‘NAMING_MASTER’;

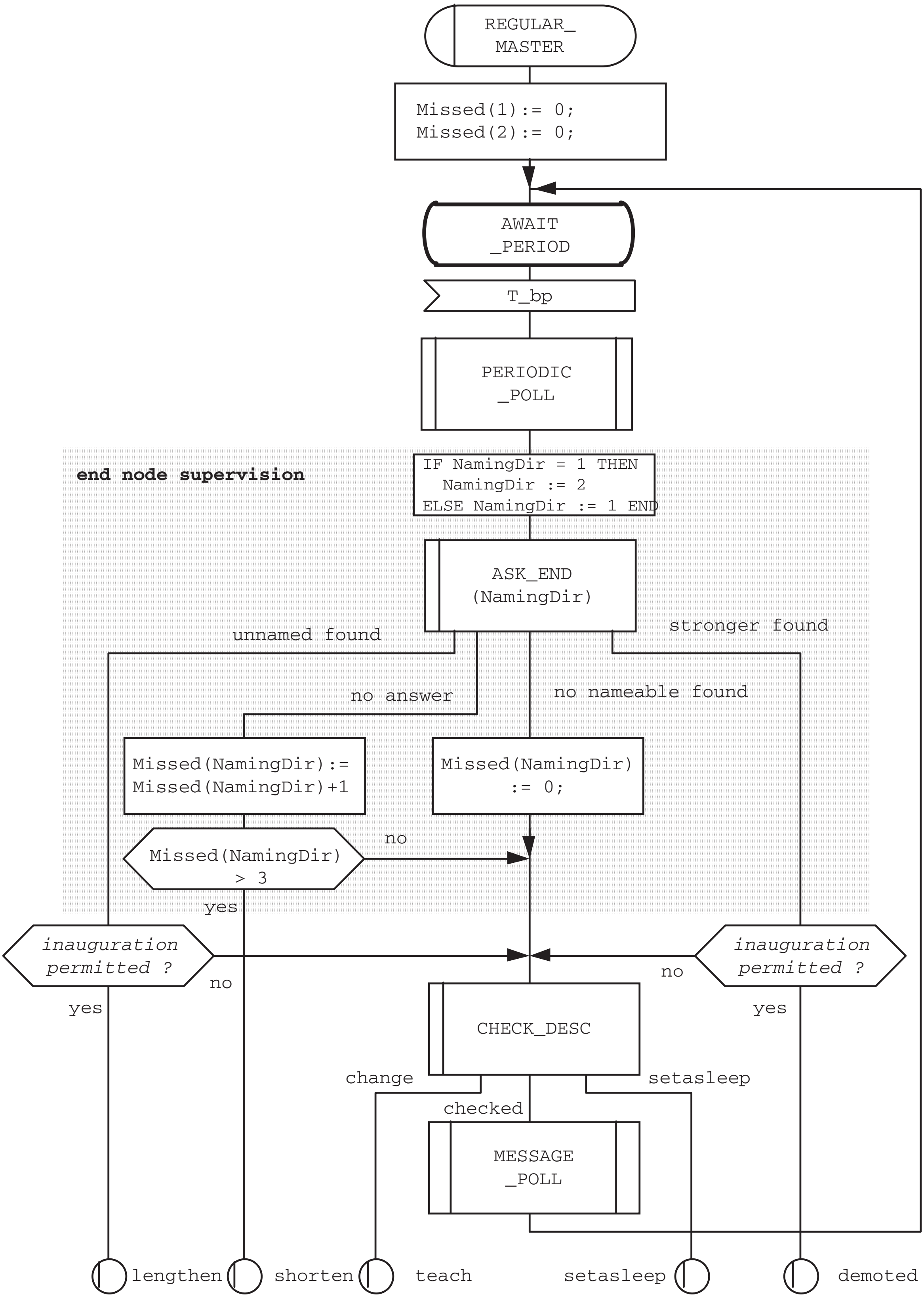
e) s'il détecte la présence d’une autre composition à nommer et que cette inauguration est activée, puis exécuter UNNAMING_MASTER pour dissoudre sa composition et passer à la macro ‘NAMING_MASTER’;

f) s'il détecte un changement de composition, puis passer à la macro ‘TEACHING_MASTER’;

g) s'il détecte une composition plus forte que la sienne avec inauguration autorisée, puis dissoudre sa composition avant de passer à la macro ‘UNNAMING_SLAVE’;

h) s'il a été mis en veille ou déconnecté par une commande de l’application, puis passer à la macro ‘UNNAMED_SLAVE’’.

NOTE L’état ‘UNNAMED_SLAVE’ met le nœud en veille lorsque tous les autres nœuds sont également dans cet état.


REGULAR_
MASTER
Missed(1):= 0;
Missed(2):= 0;
AWAIT
_PERIOD
T_bp
PERIODIC
_POLL
end node supervision
IF NamingDir = 1 THEN
NamingDir := 2
ELSE NamingDir := 1 END
ASK_END
(NamingDir)
unnamed found
stronger found
no answer
no nameable found
Missed(NamingDir):=
Missed(NamingDir)+1
Missed(NamingDir)
:= 0;
Missed(NamingDir)
> 3
no
yes
inauguration
permitted ?
no
yes
inauguration
permitted ?
no
yes
CHECK_DESC
change
setasleep
checked
MESSAGE
_POLL
lengthen
shorten
teach
setasleep
demoted

Légende

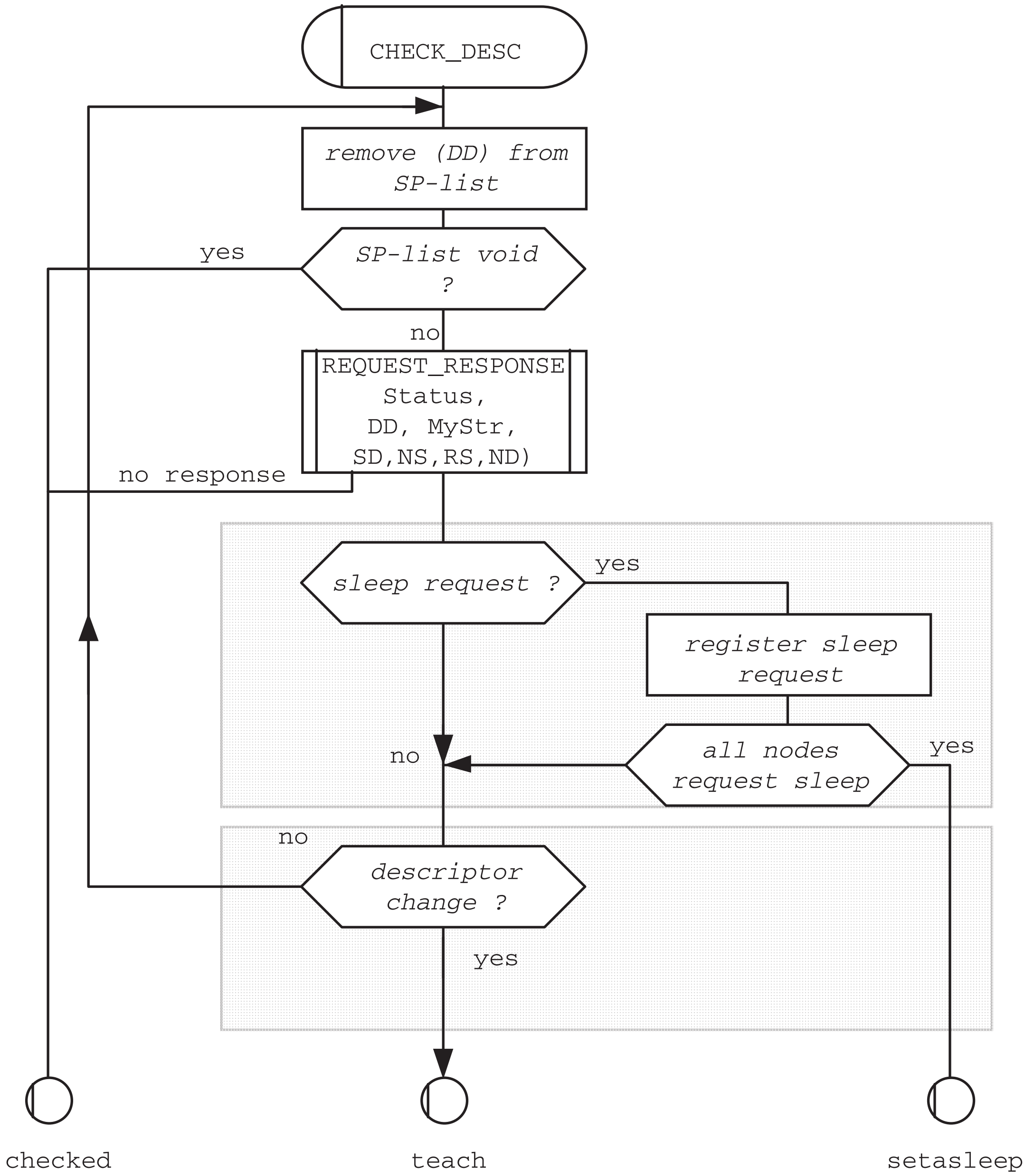
Anglais	Français
end node supervision	supervision du nœud d'extrémité
unnamed found	élément sans nom trouvé
stronger found	élément plus fort trouvé
no answer	pas de réponse
no nameable found	aucun élément à nommer trouvé
no	non
yes	oui
inauguration permitted ?	inauguration autorisée ?
change	modification
checked	contrôlé
setasleep	en veille
lengthen	rallonger
shorten	raccourcir
teach	apprendre
demoted	dégradé

Figure 98 – Macro 'REGULAR_MASTER'

5.5.4.8.10 Macro 'CHECK_DESC'

Comme le montre la Figure 99, cette macro vérifie la présence des Nœuds Intermédiaires qui ont demandé un changement de descripteur ou une mise en veille.

Le cas des Nœuds d'Extrémité est traité à part, puisqu'un Nœud d'Extrémité peut également signaler le prolongement du bus.



Légende

Anglais	Français
no	non
SP-list void	liste SP vide
no response	pas de réponse
sleep request ?	veille demandée ?
register sleep request	enregistrer demande de veille
all nodes request sleep	tous les nœuds demandent la veille
descriptor change	modification du descripteur
checked	contrôlé
teach	apprendre
setasleep	veille
yes	oui

Figure 99 – Macro CHECK_DESC

5.5.4.8.11 Macro ‘PERIODIC_POLL’

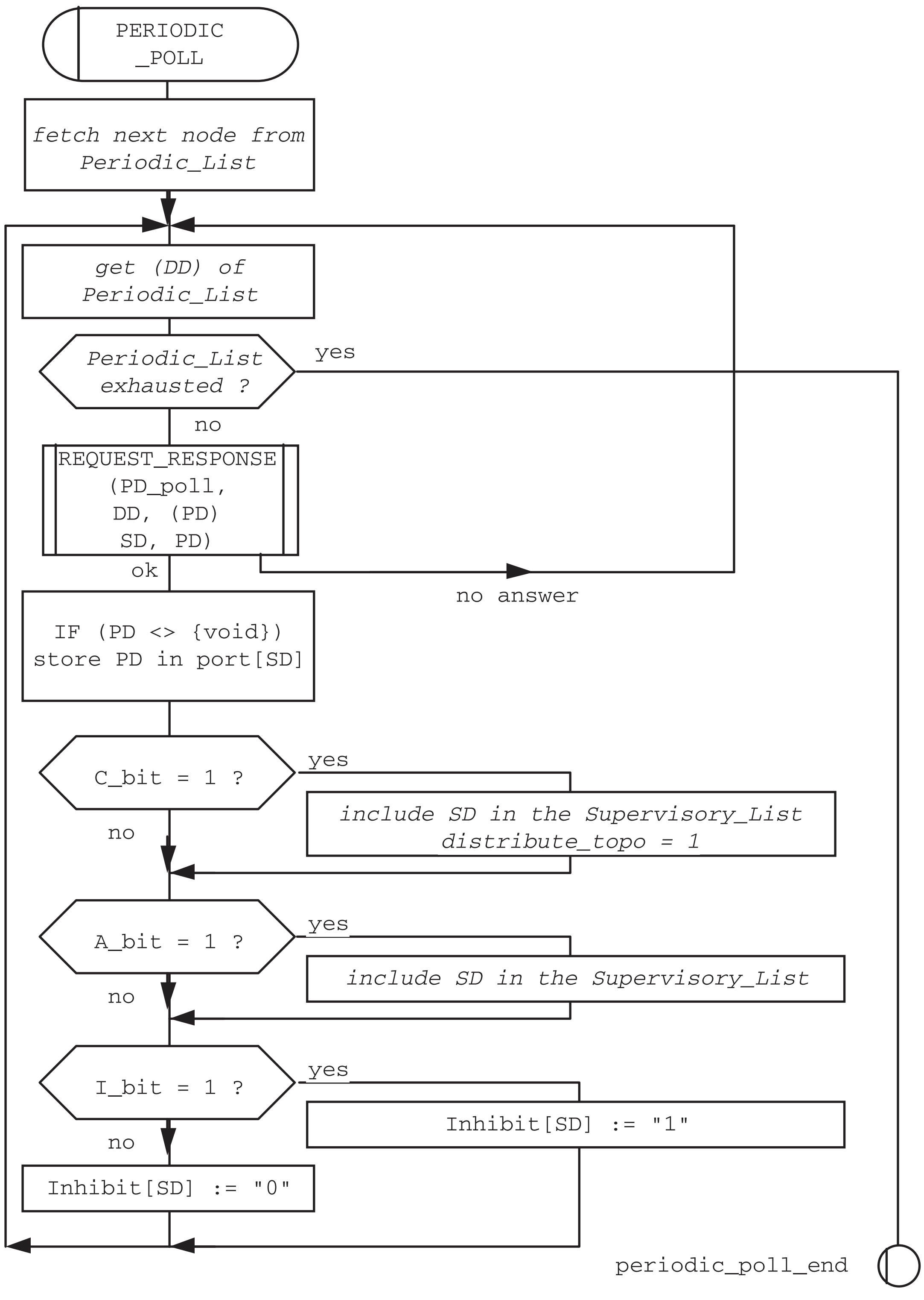
Comme le montre la Figure 100, le maître analyse les nœuds selon la Periodic_List, qui contient les adresses à interroger pendant cette période.

En interrogeant les nœuds pour les Données de Processus, le maître doit:

a) enregistrer l’existence et les Données de Processus du nœud interrogé;
b) enregistrer les changements de composition (C_bit) des nœuds qui ont modifié leur Node_Descriptor;
c) enregistrer les nœuds qui nécessitent un transfert de Données de Messagerie en activant leur A_bit;
d) enregistrer les nœuds qui bloquent l’inauguration par leur I_bit.

Le maître ne doit pas répéter la Process_Data_Request durant cette même Période de Base s’il ne reçoit pas de Process_Data_Response.

Lorsque le maître s'interroge lui-même, il doit s’envoyer une Process_Data_Request avant d'envoyer une Process_Data_Response.


PERIODIC _POLL
fetch next node from Periodic_List
get (DD) of Periodic_List
Periodic_List exhausted ?
yes
no
REQUEST_RESPONSE (PD_poll, DD, (PD) SD, PD)
ok
no answer
IF (PD <> {void}) store PD in port[SD]
C_bit = 1 ?
yes
include SD in the Supervisory_List distribute_topo = 1
no
A_bit = 1 ?
yes
include SD in the Supervisory_List
no
I_bit = 1 ?
yes
Inhibit[SD] := "1"
no
Inhibit[SD] := "0"
periodic_poll_end

Légende

Anglais	Français
fetch next node from Periodic_List	extraire le nœud suivant de Periodic_List
get (DD) of Perdiodic_List	obtenir (DD) de Periodic_List
Periodic_List exhausted ?	Periodic_List épuisée ?
yes	oui
no	non
no answer	pas de réponse
include SD in the Supervisory_List distribute_topo = 1	inclure SD dans la Supervisory_List distribute_topo = 1
include SD in the Supervisory_List	inclure SD dans la Supervisory_List

Figure 100 – Macro PERIODIC_POLL

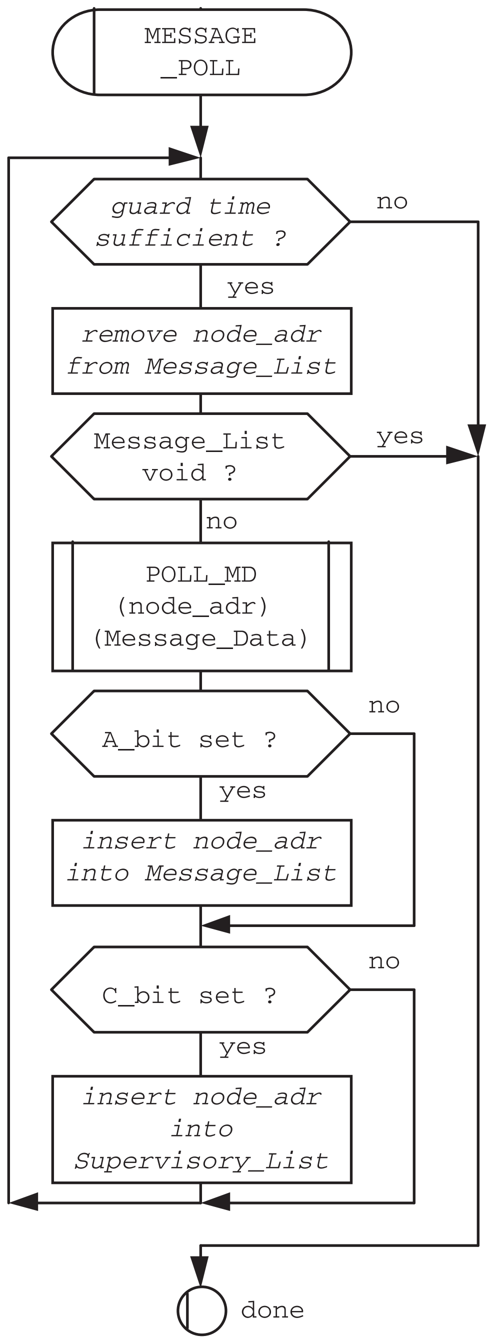
5.5.4.8.12 Macro 'MESSAGE_POLL'

Comme le montre la Figure 101, le maître doit envoyer les Données de Messagerie s'il dispose de suffisamment de temps avant la Phase Périodique suivante.

Le maître ne doit pas répéter la Message_Data_Request durant cette même Période de Base s'il ne reçoit pas de Message_Data_Response.

Lorsque le maître s'interroge lui-même, il doit s'envoyer une Message_Data_Request avant d'envoyer une Message_Data_Response.

Le nœud doit quitter cet état s'il ne reste pas suffisamment de temps pour envoyer une trame complète avant le début de la phase périodique suivante. Il doit alors revenir à l'état AWAIT_PERIOD.



Légende

Anglais	Français
guard time sufficient ?	temps de garde suffisant ?
yes	oui
no	non
remove node_adr from Message_List	retirer node_adr de Message_List
Message_List void ?	Message_List vide ?
A_bit set ?	A_bit défini ?
insert node_adr into Message_List	insérer node_adr dans Message_List
C_bit set ?	C_bit activé ?
insert node_adr intoSupervisory_List	insérer node_adr dans Supervisory_List
done	terminé

Figure 101 – Macro MESSAGE_POLL

5.5.4.9 Etats Esclave

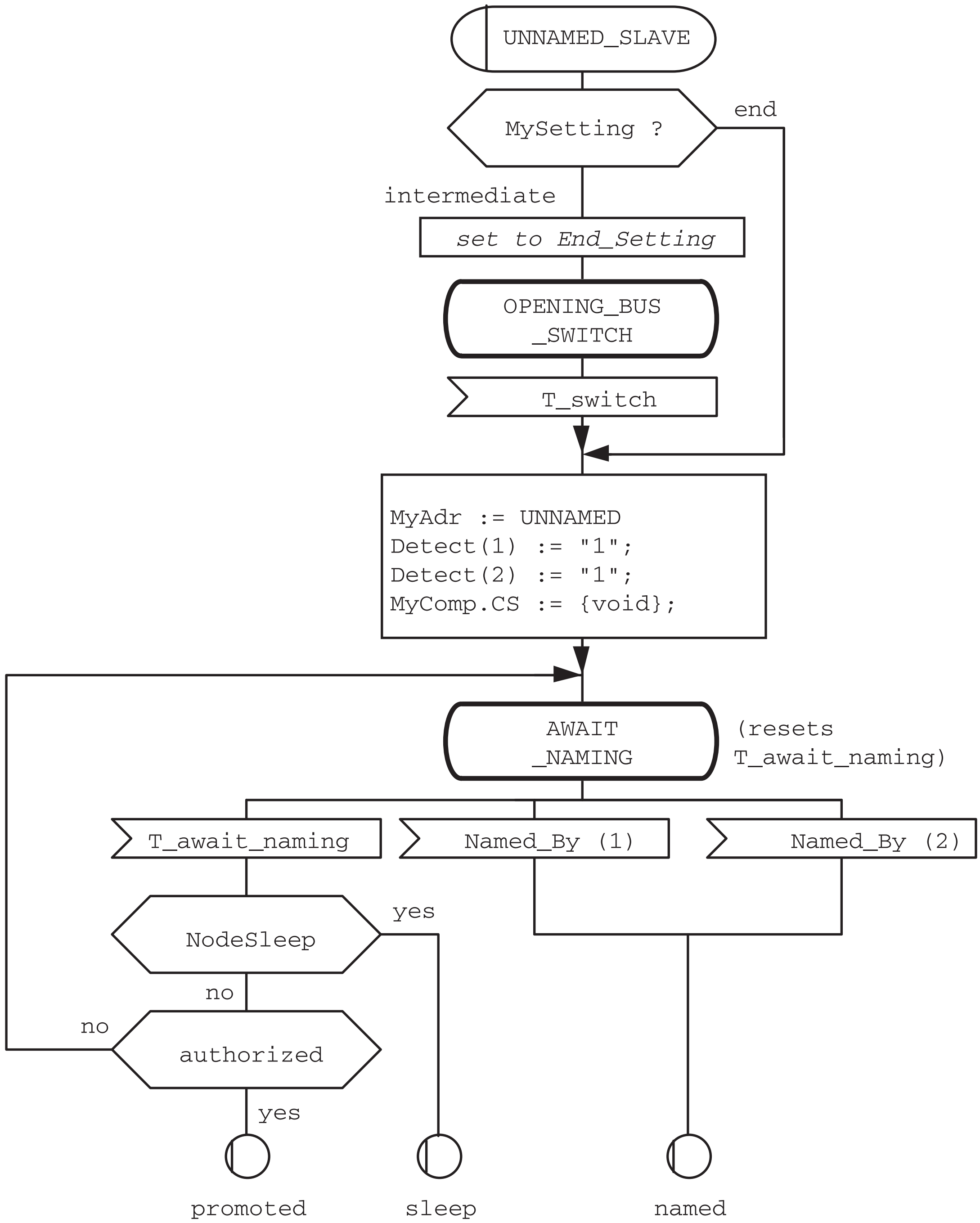
5.5.4.9.1 Macro UNNAMED_SLAVE

Comme le montre la Figure 102, un nœud doit se mettre en position End_Setting, avec ses deux Canaux Auxiliaires à l'écoute, et attendre une nomination à l’état AWAIT_NAMING.

En entrant dans l’état AWAIT_NAMING, le nœud doit redémarrer la temporisation T_await_naming et attendre:

a) un temporisateur T_await_naming
   - s’il a reçu une commande NodeSleep de la part de l’application, basculer vers Intermediate_Setting, puis passer à l'état basse puissance NODE_SLEEP;
   - s’il est configuré comme nœud faible, passer à la macro NAMING_MASTER; ou
   - sinon, retourner à UNNAMED_SLAVE;

b) un signal NamedBy du Canal Auxiliaire,
   - verrouille son Canal Principal dans la direction dans laquelle il a été nommé,
   - passe à l’état NAMED_SLAVE.



Légende

Anglais	Français
(resets T_await_naming)	(réinitialise T_await_naming)
yes	oui
no	non
authorized	autorisé
promoted	promu
sleep	veille
named	nommé

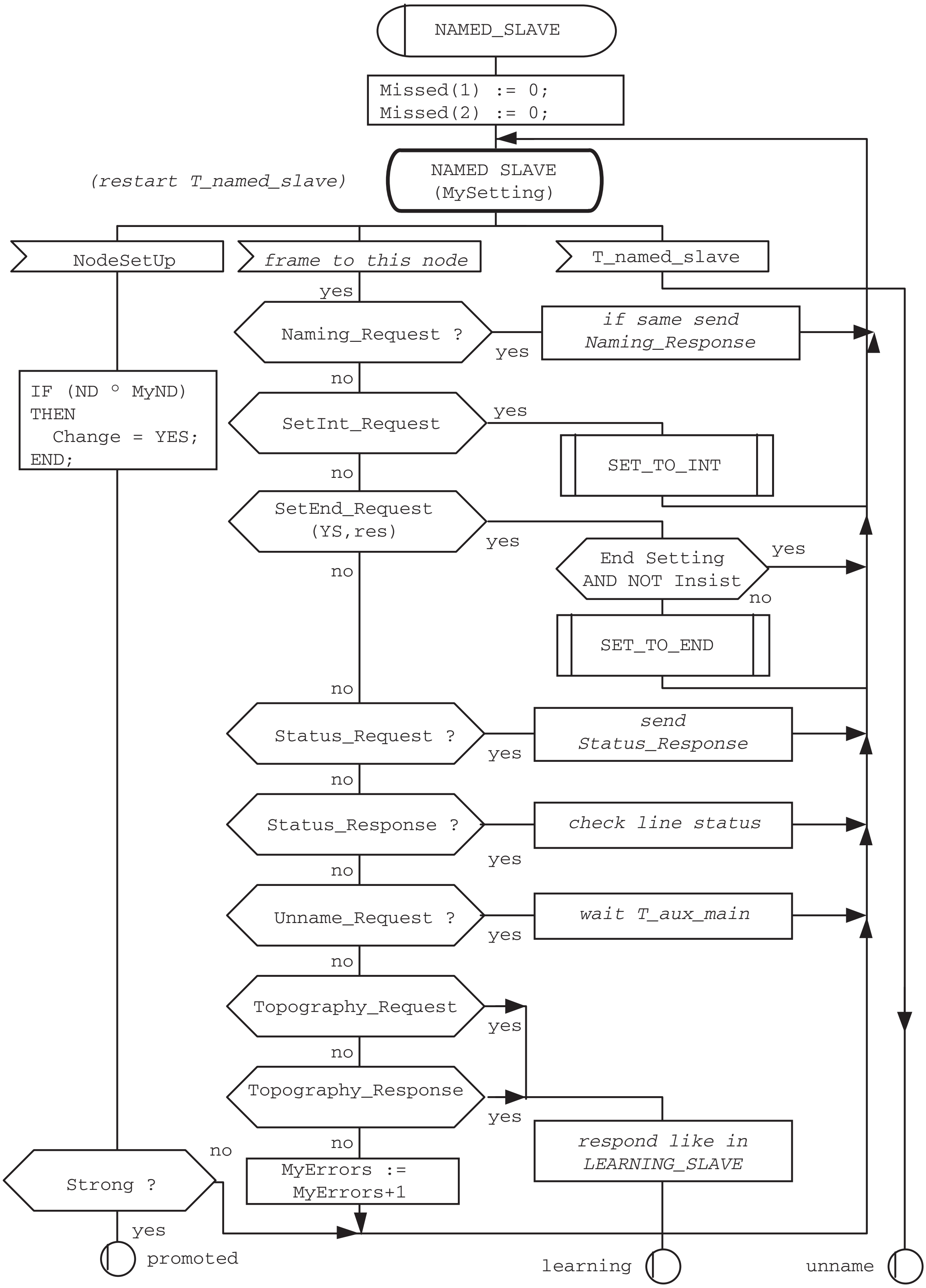
Figure 102 – États UNNAMED_SLAVE

5.5.4.9.2 Macro NAMED_SLAVE

Comme le montre la Figure 103, un nœud dans cet état a été nommé et peut passer de End_Setting à Intermediate_Setting (ou inversement) à l'intérieur de cet état en réponse aux requêtes du maître.

En entrant dans l'état NAMED_SLAVE, le nœud doit réinitialiser le temporisateur T_named_slave et attendre:

a) un changement de Node_Descriptor:
   - s'il a été promu nœud fort, aller à la macro NAMING_MASTER; ou
   - sinon, activer son 'C_bit" et retourner à NAMED_SLAVE;
b) une temporisation T_named_slave:
   - passer à UNNAMED_SLAVE;
c) Naming_Request:
   - envoyer Naming_Response au maître;
d) SetInt_Request:
   - envoyer SetInt_Response au maître, passer à Intermediate_Setting s'il n'est pas déjà dans cette position;
e) SetEnd Request:
   - envoyer SetEnd_Response au maître, passer à End_Setting s'il n'est pas déjà dans cette position;
f) Status_Request:
   - diffuser Status_Response;
g) Status_Response:
   - enregistrer l'état pour vérifier la redondance de ligne;
h) Unname_Request:
   - attendre T_aux_main (pour permettre à tous les nœuds de recevoir l'une des trois Unname_Requests);
   - mettre le seuil du temporisateur T_await_naming à sa plus haute valeur; et
   - retourner à l'état UNNAMED_SLAVE;
i) Topography Request, Topography Response:
   - répondre comme dans LEARNING_SLAVE et aller à LEARNING_SLAVE;
j) sinon:
   - incrémenter le compteur d'erreurs.


NAMED_SLAVE
Missed(1) := 0;
Missed(2) := 0;
(restart T_named_slave)
NAMED SLAVE
(MySetting)
NodeSetUp
frame to this node
T_named_slave
yes
Naming_Request ?
if same send
Naming_Response
yes
no
IF (ND ° MyND)
THEN
Change = YES;
END;
SetInt_Request
yes
SET_TO_INT
no
SetEnd_Request
(YS,res)
yes
End Setting
AND NOT Insist
yes
no
SET_TO_END
no
no
Status_Request ?
send
Status_Response
yes
no
Status_Response ?
check line status
yes
no
Unname_Request ?
wait T_aux_main
yes
no
Topography_Request
yes
no
Topography_Response
yes
respond like in
LEARNING_SLAVE
no
no
Strong ?
MyErrors :=
MyErrors+1
yes
promoted
learning
unname

Légende

Anglais	Français
(restart T_named_slave)	(redémarrer T_named_slave)
frame to this node	trame vers ce nœud
if same send Naming_Response	si le même envoie Naming_Response
yes	oui
no	non
Strong ?	Fort ?
respond like in LEARNING_SLAVE	répondre comme dans LEARNING_SLAVE
promoted	promu
learning	apprentissage
unname	sans nom

Figure 103 – Etats NAMED_SLAVE

NOTE 1 Dans la phase de nomination, la supervision du bus a lieu avec les Status_Responses, sans considérer les Nœuds d'Extrémité.

NOTE 2 Un nœud ignore Message_Data_Requests, Message_Data_Response, Process_Data_Requests et Process_Data_Responses, Presence_Requests et Presence_Responses dans cet état.

5.5.4.9.3 Macro LEARNING_SLAVE

Comme le montre la Figure 104, le nœud reçoit l'information de Topographie de tous les autres nœuds tout en supervisant le bus. Le nœud ne change ni sa position ni son nom.

En entrant dans l'état LEARNING_SLAVE, le nœud doit réinitialiser le temporisateur T_learning_slave et attendre:

a) un changement de Node_Descriptor:
   - s'il a été promu nœud fort, aller à l'état NAMING_MASTER; ou
   - sinon, il doit activer son C_bit;

b) une temporisation T_learning_slave:
   - passer à UNNAMED_SLAVE sans changer la valeur du temporisateur T_await_naming;

c) Unname_Request:
   - attendre T_aux_main (pour permettre à tous les nœuds de recevoir l'une des trois Unname_Requests);
   - définir le seuil du temporisateur T_await_naming à sa plus haute valeur T_await_max; et
   - retourner à l'état UNNAMED_SLAVE;

d) Status_Request:
   - envoyer Status_Response au maître;

e) Status_Response:
   - enregistrer l'état;

f) Topography_Request:
   - diffuser sa Topography_Response;

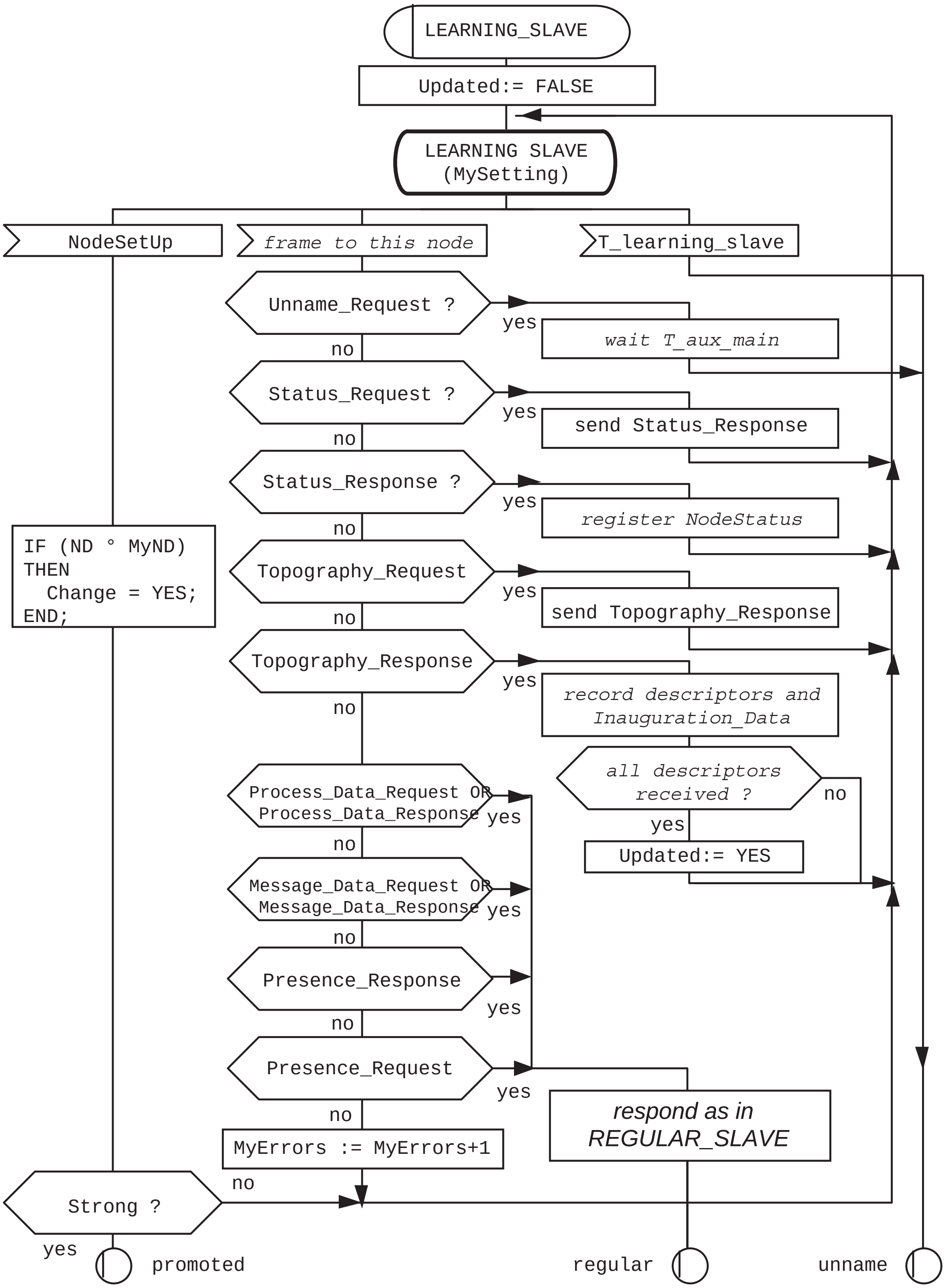
g) Topography_Response:
   - mettre à jour sa topographie, vérifier s'il est en possession d'une Topographie complète, toutes les Topography_Requests ayant été reçues avec le même Master_Topo et si oui, mettre Updated:= TRUE;

h) Presence_Request, Presence_Response, Process_Data_Request, Process_Data_Response, Message_Data_Request, Message_Data_Response

- répondre comme s'il était en REGULAR_SLAVE et passer à REGULAR_SLAVE;

i) sinon:

- incrémenter le compteur d'erreurs.



Légende

Anglais	Français
frame to this node	trame vers ce nœud
yes	oui
no	non
wait T_aux_main	attendre T_aux_main
send _Status_Response	envoyer Status_Response
register NodeStatus	enregistrer NodeStatus
send Topography_Response	envoyer Topography_Response
record descriptors and Inauguration_Data	enregistrer les descripteurs et Inauguration_Data
all descriptors received ?	tous les descripteurs reçus ?
Respond as in REGULAR_SLAVE	Répondre comme dans REGULAR_SLAVE
Strong ?	Fort ?
promoted	promu
regular	régulier
unname	sans nom

Figure 104 – Macro LEARNING_SLAVE

NOTE Dans cette phase d'apprentissage, la supervision du bus se fait par les Topography_Responses, sans considérer les Nœuds d'Extrémité.

5.5.4.9.4 Macro REGULAR_SLAVE

Comme le montre la Figure 105, il s'agit de l'état de fonctionnement normal d'un nœud, dans lequel il envoie et reçoit des Données de Processus et des Données de Messagerie et signale des événements en positionnant les bits indicateurs dans ses réponses. Un nœud supervise l'activité des Nœuds d'Extrémité grâce à deux temporisateurs.

Dans l'état REGULAR_SLAVE, le nœud doit attendre:

a) un changement de Node_Descriptor:
   - s'il a été promu nœud fort, aller à l'état NAMING_MASTER; ou
   - sinon, activer son C_bit;
b) une temporisation T_bus_check pour chaque Nœud d'Extrémité qu'il supervise:
   - passer à UNNAMED_SLAVE sans changer la valeur de début de T_await_naming;
c) Presence_Request:
   - diffuser Presence_Response (le nœud doit ignorer Presence_Request s'il ne s'agit pas d'un Nœud d'Extrémité);
d) Presence_Response:
   - redémarrer le temporisateur T_bus_check correspondant;
e) Status_Request:
   - envoyer Status_Response au maître;
f) Unname_Request:
   - attendre T_aux_main (pour permettre à tous les nœuds de recevoir l'une des trois Unname_Requests);
   - définir le seuil du temporisateur T_await_naming à sa plus haute valeur; et
   - passer à UNNAMED_SLAVE;
g) Process_Data_Request sans Données de Processus:

- si le nœud est à jour, envoyer une Process_Data_Response, en lisant les données sur son port émetteur;
- sinon, envoyer une Process_Data_Response vide;

h) (option) Process_Data_Request avec Données de Processus:

- si Updated = TRUE, écrire ces données dans le port destinataire destiné aux données du maître direct, et envoyer une Process_Data_Response, en lisant les données sur son port émetteur;
- Sinon, les ignorer et envoyer une Process_Data_Response vide;

i) Process_Data_Response:

- si Updated = TRUE, écrire les données dans le port destinataire correspondant à l'adresse source;
- sinon les ignorer;

j) Message_Data_Request:

- si la file d'émission est vide, envoyer une Message_Data_Response vide (link_data_size = 0) au maître; ou
- envoyer une Message_Data_Response avec un paquet de Données de Messagerie tiré de sa file d'émission;

k) Message_Data_Response:

- enregistrer les Données de Messagerie entrantes dans sa Receive_Queue, s'il y a de la place, sinon, les ignorer (voir 5.6.3);

l) Topography_Request ou Topography_Response:

- mettre Update = FALSE, répondre comme indiqué dans LEARNING_SLAVE et passer à cet état;

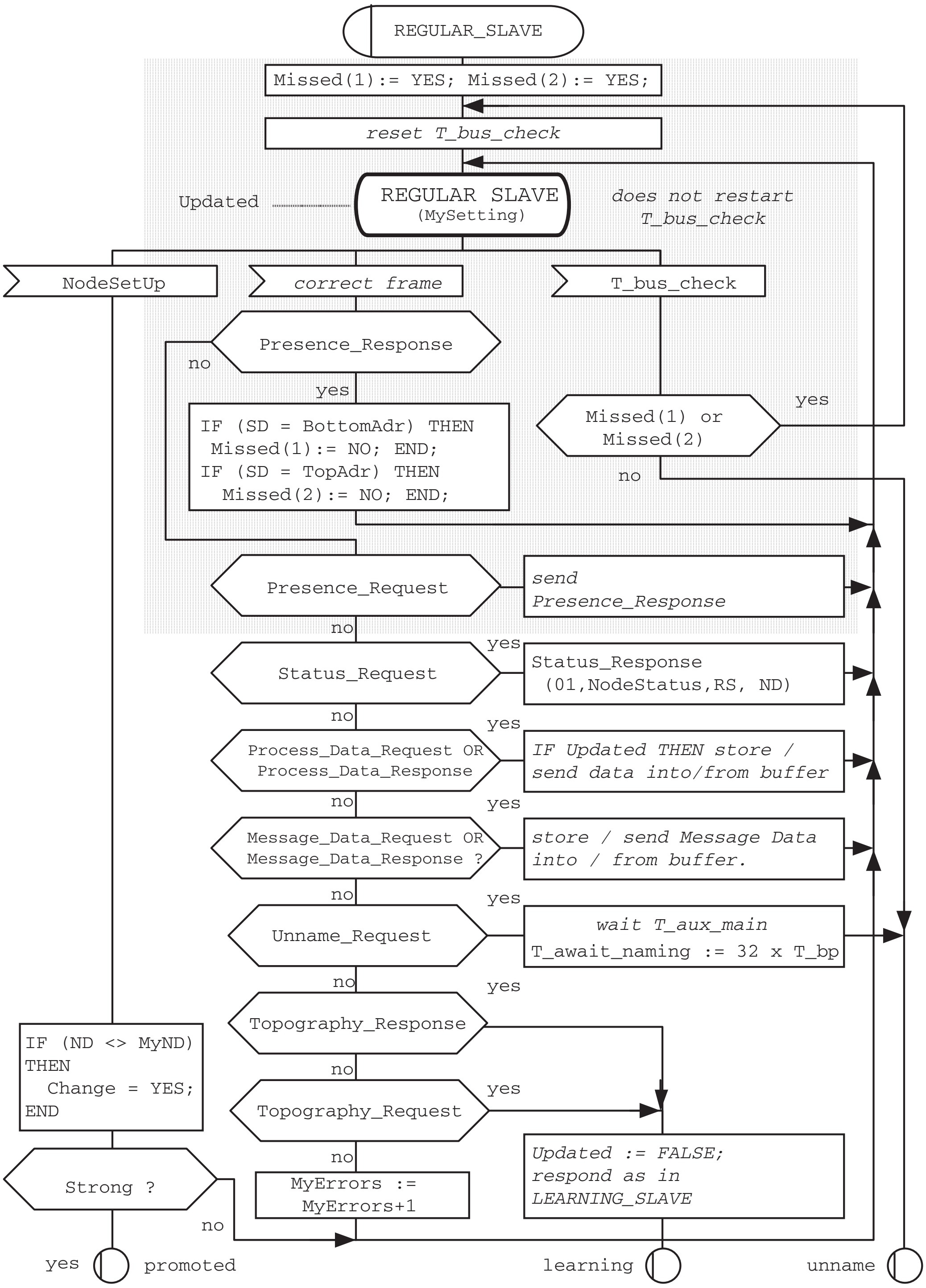
m) sinon:

- incrémenter son compteur d'erreurs et retourner à LEARNING_SLAVE.

NOTE 1 Le temporisateur T_bus_check peut être mis en œuvre avec un seul temporisateur et des compteurs.

NOTE 2 Aucun temporisateur n'est associé à l'état REGULAR_SLAVE, ni redémarré lors de l'entrée dans cet état, cette fonction étant exercée par T_bus_check.

NOTE 3 La supervision des Nœuds d'Extrémité peut également être effectuée individuellement avec un temporisateur T_bus_check pour chaque nœud d'extrémité.


REGULAR_SLAVE
Missed(1):= YES; Missed(2):= YES;
reset T_bus_check
Updated
REGULAR SLAVE
(MySetting)
does not restart
T_bus_check
NodeSetUp
correct frame
T_bus_check
Presence_Response
no
yes
IF (SD = BottomAdr) THEN
Missed(1):= NO; END;
IF (SD = TopAdr) THEN
Missed(2):= NO; END;
Missed(1) or
Missed(2)
yes
no
Presence_Request
send
Presence_Response
Status_Request
Status_Response
(01,NodeStatus,RS, ND)
Process_Data_Request OR
Process_Data_Response
IF Updated THEN store /
send data into/from buffer
Message_Data_Request OR
Message_Data_Response ?
store / send Message Data
into / from buffer.
Unname_Request
wait T_aux_main
T_await_naming := 32 x T_bp
Topography_Response
Topography_Request
IF (ND <> MyND)
THEN
Change = YES;
END
Strong ?
MyErrors :=
MyErrors+1
Updated := FALSE;
respond as in
LEARNING_SLAVE
promoted
learning
unname

Légende

Anglais	Français
reset T_bus_check	réinitialiser T_bus_check
Updated	Mis à jour
does not restart T_bus_check	ne pas redémarrer Y_bus_check
correct frame	trame correcte
no	non
yes	oui
Missed(1) or Missed(2)	Missed(1) ou Missed(2)
send Presence_Response	envoyer Presence_Response
IF Updated THEN store / send data into/from buffer	SI Mis à jour ALORS enregistrer/envoyer données dans/à partir du tampon
store / send Message data into/from buffer	enregistrer/envoyer données de message dans/à partir du tampon
Strong	fort
wait T_aux_main	attendre T_aux_main

Figure 105 – Macro REGULAR_SLAVE

5.5.4.10 Temporisations

Le Tableau 17 donne les valeurs préconisées des temporisations (± 20 %).

Tableau 17 – Valeurs des constantes de temps

Nom de la constante de temps	Valeur	Usage
T_await_naming	1) T_await_min = 1,0 ms + T_switch	Maître renommant sa composition
	2) ((63 – MyAdr) + 0,5) × T_bp	Nœuds dans la direction 1 du maître
	3) (MyAdr-1) × T_bp	Nœuds dans la direction 2 du maître
	4) T_await_max = 32 × T_bp	Nœuds initialisés ou explicitement sans nom
T_aux_main	1,0 ms	Temps de commutation du canal auxiliaire au canal principal (ou l'inverse)
T_await_response	1,756 ms	Temps d'attente du maître pour une Trame-Esclave. Ce temps tient compte de la trame la plus longue car les contrôleurs HDLC ne peuvent signaler que la fin d'une trame.
T_bp	25,0 ms	Période de Base (en fonctionnement normal)
T_naming_master	15,0 ms	période de nomination (inauguration)
T_named_slave	15,0 ms	supervision du maître pendant la phase de nomination
T_learning_slave	15,0 ms	supervision du maître pendant la phase d'apprentissage
T_bus_check (1) T_bus_check(2)	6,5 × T_bp	Supervision des Nœuds d'Extrémité en fonctionnement normal.
T_detecting	2 × T_naming_master	Intervalle entre Detect_Request (le cas échéant) pendant l'inauguration
	2 × T_bp	Intervalle entre Presence_Request et Detect_Request (le cas échéant) pendant le fonctionnement normal.
T_detecting_response	1 047 ms	Temps pendant lequel le Nœud d'Extrémité attend une Detect_Response.
T_new_inaug	n × T_bp	Temps minimal entre deux inaugurations consécutives. n est déterminé par l'application
MAXLOST	50	T_detecting × MAXLOST = temps après lequel un Nœud d'Extrémité considère qu'il n'y a plus d'autre composition présente.
T_switch	10,0 ms	Temps par défaut de fermeture ou d'ouverture des relais.

5.6 Interface Couche de Liaison

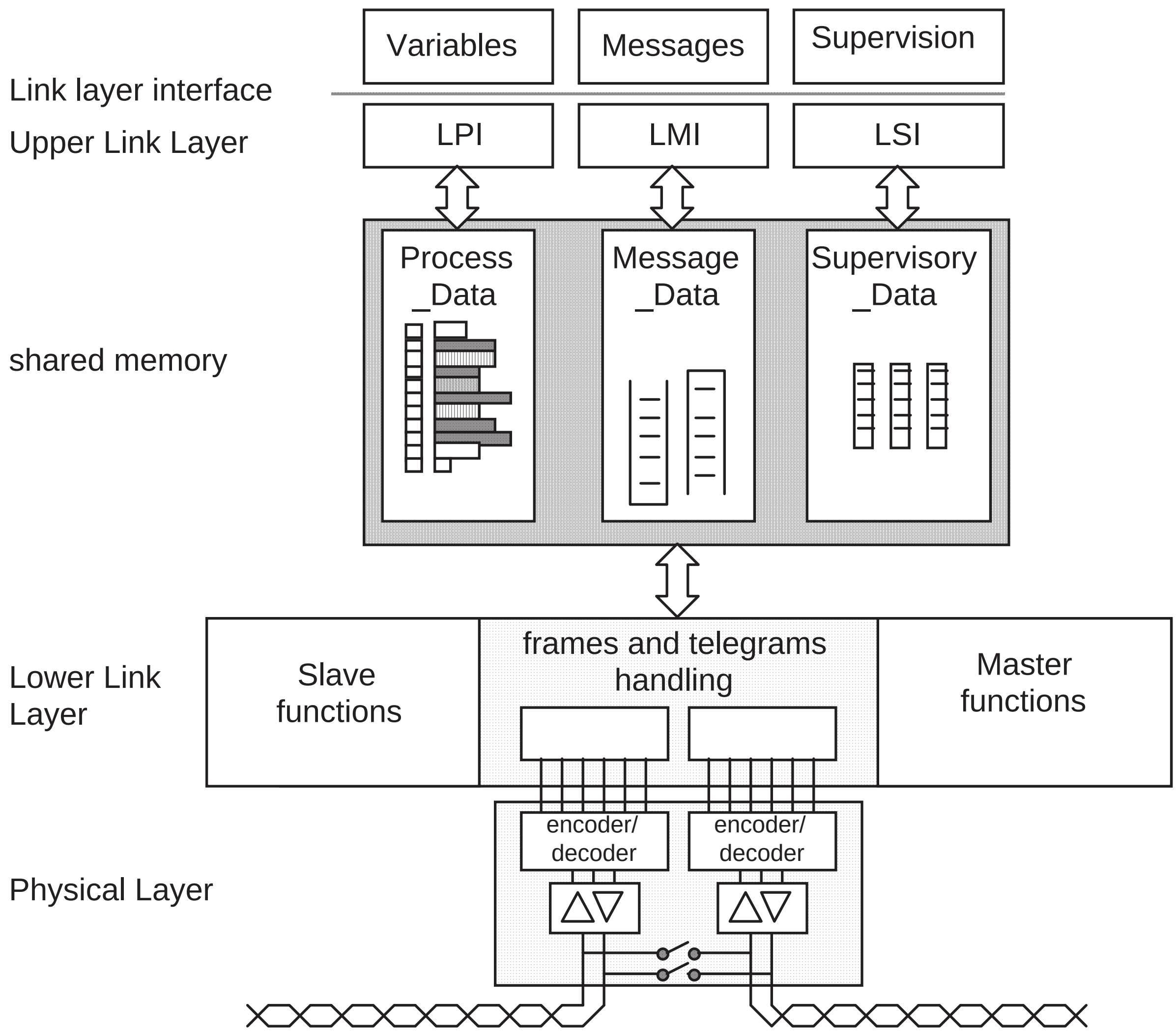
5.6.1 Organisation de la Couche de Liaison

L'interface Couche de Liaison fournit trois services, comme illustré à la Figure 106:

a) l’interface de liaison de Données de Processus (LPI), qui est utilisée par le Service de Variables, est spécifiée dans l'Article 6 (Protocoles en Temps Réel). Seuls les paramètres spécifiques à WTB sont spécifiés dans la présente norme;

b) l’interface de liaison de Données de Messagerie (LMI), qui est utilisée par les services de Messages, est spécifiée dans l'Article 6 (Protocoles en Temps Réel). Seuls les paramètres spécifiques à WTB sont spécifiés dans la présente norme;

c) l’interface de Supervision de Liaison (LSI), qui permet la configuration de la Couche de Liaison et la supervision du bus, est spécifique au WTB et est spécifiée dans la présente norme.



Légende

Anglais	Français
Link layer interface	Interface de couche de liaison
shared memory	mémoire partagée
Upper Link Layer	Couche de liaison supérieure
Lower Link Layer	Couche de liaison inférieure
Slave functions	Fonctions esclaves
frames and telegrams handling	traitement des trames et des télégrammes
Master functions	Fonctions maîtres
Physical Layer	Couche physique
encoder/decoder	codeur/décodeur

Figure 106 – Organisation de la Couche de Liaison

5.6.2 Interface de liaison de Données de Processus

5.6.2.1 Généralités

L'interface de Données de Processus entre la couche de liaison et les couches supérieures est une mémoire partagée appelée Traffic_Store, à laquelle peuvent accéder simultanément le bus et l'application.

Cette mémoire est structurée en un certain nombre de Ports qui contiennent chacun exactement une trame d'émission ou de réception de Données de Processus.

Chaque Port est identifié dans un nœud par un indicatif Traffic_Store et une adresse port de 12 bits.

La mise en œuvre des Ports ne fait pas partie de la normalisation.

L'Article 6 (Protocoles en Temps Réel) spécifie l'accès aux ports.

5.6.2.2 Spécifique au WTB

La mémoire Traffic_Store du WTB doit comprendre jusqu'à 64 Ports, de 1 024 bits maximum chacun.

Dans tous les cas:

- chaque nœud doit comporter un port émetteur pour diffuser ses Données de Processus, les six bits de poids fort d'adresse de port étant 000000'B et les six bits de poids faible d'adresse de port étant l'adresse de ce nœud;
- chaque nœud doit comporter un port destinataire pour recevoir les Données de Processus de chacun des autres nœuds possibles sur le bus, les six bits de poids fort d'adresse de port étant à '000000'B et les six bits de poids faible d'adresse de port étant l'adresse du nœud émetteur.

Dans les applications où le maître inclut des Données de Processus dans ses Process_Data_Request, destinées uniquement à l'esclave interrogé:

- le maître doit comporter un port émetteur pour envoyer des Données de Processus à chaque esclave possible, les six bits de poids fort étant à '000010'B et les six bits de poids faible de l'adresse de port étant l'adresse du nœud destinataire;

- chaque esclave doit mettre en œuvre un port destinataire pour recevoir les Données de Processus du maître, les bits de poids fort étant à ‘000010’B et les six bits de poids faible d’adresse de port étant l’adresse de ce nœud.

Un nœud ne doit pas accepter les Données de Processus d'un nœud qui n’a pas été inclus dans la Topographie reçue ou dont il ne peut interpréter le descripteur reçu dans la Topographie.

Un nœud peut accepter des Données de Processus d'un autre nœud dont il connaît le Node_Type mais dont la Node_Version est différente de celle qu’il connaît. Cependant, il doit décoder ces données selon la version inférieure des deux Node_Versions.

Un nœud ne peut changer le format de sa Process_Data_Response sans avoir reçu une Topographie comprenant sa nouvelle Node_Key.

Il est recommandé de réserver les deux premiers octets des Données de Processus pour identifier le type de trame (Node_Type + Node_Version = Node_Key) comme protection additionnelle.

5.6.3 Interface de liaison de Données de Messagerie

5.6.3.1 Généralités

L’interface de liaison de Données de Messagerie (LMI) définie dans l'Article 6 (Protocoles en Temps Réel) fournit des services d'envoi de trames de Données de Messagerie et de récupération des trames de Données de Messagerie reçues. De plus, elle offre des services de confirmation d’émission et d’indication de réception.

L’interface de liaison de Données de Messagerie fournit le service de base, sur lequel tous les protocoles supérieurs sont construits:

a) la couche réseau permet l'acheminement à travers les fonctions de réseau et de répertoire;
b) le protocole de transport permet le contrôle de bout en bout semi-duplex de messages;
c) la couche session couple les messages pour fournir un Appel de Procédure à Distance;
d) la couche de présentation unifie la présentation des données;
e) l'interface application fournit les interfaces client et serveur.

5.6.3.2 Taille des paquets

Le champ ‘link_data_size’ d'un paquet vide doit être nul.

Le champ ‘link_data_size’ ne doit pas être supérieur à 128.

5.6.3.3 Protocol_Type

Le Protocol_Type pour les Protocoles en Temps Réel doit être indiqué par le champ link_control mis à ‘00xxx111’B (Message_Data_Response).

5.6.3.4 Protocole de transport de messages

Un nœud WTB ne doit pas annoncer une taille de paquet de plus de 124 octets dans sa Connect_Request.

En répondant à une Connect_Request, un nœud doit préciser dans sa Connect_Response une taille de paquet égale à 124 octets ou la taille proposée du paquet, la plus petite étant retenue.

5.6.4 Interface de gestion de la Couche de Liaison

5.6.4.1 Généralités

L'interface de gestion de la Couche de Liaison est propre au WTB.

Elle fournit les services généraux pour la configuration et le contrôle de la Couche de Liaison, ainsi que pour la signalisation d'événements.

Les paragraphes qui suivent n'impliquent pas une mise en œuvre particulière. Toute interface qui fournit la même sémantique est autorisée.

Le format des paramètres des procédures d’interface qui suivent n'est pas spécifié. Cependant, l'Article 6 de la norme TNM (Train Network Management) propose un format de message qui est préconisé pour le format des paramètres.

5.6.4.2 Procédures d’interface

Les procédures de cette interface comportent le préfixe ls_t (supervision de liaison, WTB).

5.6.4.2.1 Type LS_T_RESULT

Les résultats possibles sont du type LS_T_RESULT:

Constante	Code	Signification
L_OK	0	succès
L_BUSY	1	essayer à nouveau ultérieurement
L_CALLING_SEQUENCE	2	séquence de commande erronée
L_MISSING_UDF	3	fonction utilisateur inconnue
L_CONFIGURATION_INVALID	4	Topographie ou liste de nœuds invalide

5.6.4.2.2 Constantes LS_T_STATE

Les constantes suivantes indiquent l'état principal actuel du nœud:

Constante	Code	Signification
LS_INITIALIZED	0	nœud à l'état UNCONFIGURED
LS_CONFIGURED	1	nœud à l'état CONFIGURED
LS_READY_TO_NAME	2	nœud à l'état NAMING_MASTER
LS_READY_TO_BE_NAMED	3	nœud à l'état UNNAMED_SLAVE
LS_INHIBITED	4	inauguration du nœud bloquée
LS_REGULAR_STRONG	5 6	nœud à l'état REGULAR_MASTER (fort) ou TEACHING_MASTER
LS_REGULAR_SLAVE	7	nœud à l'état REGULAR_SLAVE ou TEACHING_MASTER
LS_REGULAR_WEAK		nœud à l'état REGULAR_MASTER (faible) ou TEACHING_MASTER

NOTE Un nœud ne peut indiquer qu'il est en état de veille.

5.6.4.3 Génération de rapports

5.6.4.3.1 Procédure ls_t_Report

Action	Signale à l'utilisateur un changement dans la couche de liaison. Cette procédure est appelée par la couche de liaison et doit faire l'objet d'un abonnement préalable (voir: ls_t_Configure).	
Syntaxe	`Typedef LS_T_RESULT (* ls_t_Report) (ls_report)`	
Entrée	ls_report	l'un des codes de rapport LR_REPORT

5.6.4.3.2 Constantes LR_REPORT

La valeur des codes de rapport doit être comme suit:

Constante	Code	Signification
LR_CONFIGURED	16	La couche de liaison est configurée
LR_STRONG	17	Le nœud est le maître opérationnel
LR_SLAVE	18	Le nœud est un esclave opérationnel
LR_PROMOTED	19	Le nœud passe de maître faible à maître fort
LR_NAMING_SUCCESSFUL	20	Le maître signale la fin de l'inauguration
LR_NAMED	21	Le nœud est un esclave nommé
LR_WEAK	22	Le maître devient un maître faible
LR_REMOVED	23	Le nœud est retiré de la configuration
LR_DEMOTED	24	Un maître faible a détecté un maître fort
LR_DISCONNEXION	25	Le nœud est déconnecté
LR_INHIBITED	26	Inauguration bloquée
LR_INCLUDED	27	Inclus dans la composition
LR_LENGTHENING	28	Le maître a détecté un prolongement du train
LR_DISRUPTION	29	Un nœud a détecté la perte du Nœud d'Extrémité
LR_MASTER_CONFLICT	30	Un maître fort a détecté un autre maître fort
LR_NAMING_FAILED	31	Défaillance lors du nommage
LR_NEW_TOPOGRAPHY	32	Arrivée d'une nouvelle Topographie
LR_NODE_STATUS	33	État d'un nœud modifié
LR_POLL_LIST_OVF	34	Fonctionnement partiel
LR_ALLOWED	35	Inauguration autorisée

5.6.4.4 Service d'initialisation

5.6.4.4.1 Procédure ls_t_Init

Action	Initialise la couche de liaison et met les variables aux valeurs pré-définies. Suite à l'appel, la couche de liaison doit être prête à recevoir les commandes. Cette procédure ne doit être appelée qu'après la remise à zéro du matériel. Cette procédure dépend de la mise en œuvre.
Syntaxe	`LS_T_RESULT ls_t_Init (void);`

5.6.4.5 Service Reset

5.6.4.5.1 Procédure ls_t_Reset

Action	Remet la couche de liaison aux valeurs pré-définies. Suite à l'appel, la couche de liaison doit être en veille, prête à recevoir les commandes.
Syntaxe	`LS_T_RESULT ls_t_Reset (void);`

5.6.4.6 Service Configuration

5.6.4.6.1 Procédure ls_t_Configure

<table>
<tr><td>Action</td><td colspan="2">Configure la couche de liaison.
Suite à l'appel, le nœud doit être prêt à communiquer</td></tr>
<tr><td>Syntaxe</td><td colspan="2"><pre>LS_T_RESULT ls_t_Configure
    (
    Type_Configuration *   p_configuration
    );</pre></td></tr>
<tr><td>Entrée</td><td>p_configuration</td><td>Pointeur vers la structure de données de configuration suivante.</td></tr>
</table>

5.6.4.6.2 Type_NodeKey

Une structure de données de Type_NodeKey doit comprendre les éléments suivants:

Attribut	Type	Signification
node_type	UNSIGNED8	type du nœud comme défini par l'application
node_version	UNSIGNED8	version du nœud comme définie par l'application

NOTE Type_NodeKey est le Type C correspondant à la structure Node_Key (voir 5.5.2.1).

5.6.4.6.3 Type_NodeDescriptor

Une structure de données de Type_NodeDescriptor doit comprendre les éléments suivants:

Attribut	Type	Signification
node_frame_size	UNSIGNED8	taille de la trame Données de Processus en octets.
node_period	UNSIGNED8	valeur de la Node_Period en 2n multiples de la Période de Base (node_period est la valeur de n).
node_key	Type_NodeKey	voir ce type de données

NOTE Type_NodeDescriptor est le Type C correspondant à la structure Node_Descriptor (voir 5.5.2.1).

5.6.4.6.4 Type_Configuration

La structure de données Configuration doit comprendre les éléments suivants:

Attribut	Type	Signification
transmission_rate	UNSIGNED16	vitesse de transmission en kbit/s, par défaut: 1000 kbit/s
basic_period	UNSIGNED16	Période de Base en millisecondes, par défaut: 25,0 ms
fritting_disabled	UNSIGNED16	= 1 si le nettoyage des contacts est désactivé, par défaut: 0
node_descriptor	Type_NodeDescriptor	Voir 5.6.4.6.3
poll_md_when_idle	UNSIGNED8	= 1 si l'exploration en arrière-plan est activée, par défaut: 0 (voir 5.4.3.4)
sink_port_count	UNSIGNED16	Nombre maximal de ports destinataires, par défaut: 22
source_port_count	UNSIGNED16	Nombre maximal de ports émetteurs, par défaut: 1
port_size	UNSIGNED8	Longueur maximale d'un Port en octets, par défaut: 128
p_traffic_store	WORD32	Pointeur vers la mémoire de trafic, par défaut: NULL
ls_t_report	WORD32	Fonction de rappel pour le rapport, par défaut: NULL
max_number_nodes	UNSIGNED8	Nombre maximal de nœuds dont les données d'inauguration doivent être stockées, par défaut: 0
inaug_data_max_size	UNSIGNED8 (≤ 124)	Nombre maximal d'octets des données d'inauguration définies par l'application, à envoyer, par défaut: 0
s_inaug_data_size	UNSIGNED8 (≤ 124)	Nombre réel d'octets des données d'inauguration définies par l'application, à envoyer, par défaut: 0
p_inaug_data_list	WORD32	Pointeur vers la zone de données dans laquelle copier les données d'inauguration, par défaut: NULL

5.6.4.6.5 Type_Inauguration_Data

La structure de données Type_Inauguration_Data doit contenir les éléments suivants:

Attribut	Type	Signification
inaug_data_max_size	UNSIGNED8 (≤ 124)	Nombre maximal d'octets des données d'inauguration définies par l'application, stockées, par défaut: 0
nr_descriptors	UNSIGNED8	nombre de nœuds dont les données d'inauguration sont stockées, par défaut: 0 = invalide
node_descriptions	ARRAY [nr_descriptors] OF	liste des données d'inauguration définies par l'application pour chaque nœud WTB, comprenant:
node_type	WORD8	première partie de Node_Key
node_version	WORD8	deuxième partie de Node_Key
sam	BOOLEAN1	'1' si l'orientation est identique à celle du maître
rsv1	WORD1 (=0)	réservé, =0
node_address	UNSIGNED6	adresse du nœud à partir duquel les données d'inauguration ont été reçues
inauguration_data_size	UNSIGNED8	taille d'Inauguration_Data (≤ 124 octets)
inauguration_data	ARRAY[inaug_data_len] OF WORD8	données d'inauguration définies par l'application

Les node_descriptions doivent être initialisées avant de commencer l'inauguration. A la fin de l'inauguration, la table 'node_descriptions' comprend 'nr_descriptors' lignes, une pour chaque nœud.

5.6.4.7 Service Set Slave

5.6.4.7.1 Procédure ls_t_SetSlave

Action	Empêche un nœud de devenir maître.
Syntaxe	`LS_T_RESULT ls_t_SetSlave (void);`

5.6.4.8 Service Set Weak

5.6.4.8.1 Procédure ls_t_SetWeak

Action	Permet à un nœud de devenir maître faible.
Syntaxe	`LS_T_RESULT ls_t_SetWeak (void);`

5.6.4.9 Service Set Strong

5.6.4.9.1 Procédure ls_t_SetStrong

Action	Commande à un nœud de devenir maître fort.
Syntaxe	`LS_T_RESULT ls_t_SetStrong (void);`

NOTE Cette commande provoque une inauguration.

5.6.4.10 Service StartNaming

5.6.4.10.1 Procédure ls_t_StartNaming

Action	Commande au nœud de commencer une inauguration.
Syntaxe	`LS_T_RESULT ls_t_StartNaming (void);`

5.6.4.11 Service Remove

5.6.4.11.1 Procédure ls_t_Remove

Action	Commande au nœud de se retirer de la configuration et de passer à un état passif.
Syntaxe	`LS_T_RESULT ls_t_Remove (void);`

5.6.4.12 Service Inhibit

5.6.4.12.1 Procédure ls_t_Inhibit

Action	Empêche le prolongement du bus si des nœuds supplémentaires sont détectés.
Syntaxe	`LS_T_RESULT ls_t_Inhibit (void);`

5.6.4.13 Service Allow

5.6.4.13.1 Procédure ls_t_Allow

Action	Permet le prolongement d'un bus si des nœuds supplémentaires sont détectés.
Syntaxe	`LS_T_RESULT ls_t_Allow (void);`

5.6.4.14 Service SetSleep

5.6.4.14.1 Procédure ls_t_SetSleep

Action	Provoque le signalement par le nœud d'une requête de veille.
Syntaxe	`LS_T_RESULT ls_t_SetSleep (void);`

5.6.4.15 Service CancelSleep

5.6.4.15.1 Procédure ls_t_CancelSleep

Action	Provoque l'annulation par le nœud d'une requête de veille.
Syntaxe	`LS_T_RESULT ls_t_CancelSleep (void);`

5.6.4.16 Service GetStatus

5.6.4.16.1 Procédure ls_t_GetStatus

Action	Récupère l'état de la couche physique et de la couche de liaison.	

Syntaxe	`LS_T_RESULT`
	
	`Type_WTBStatus*`
		`ls_t_GetStatus`
		`(`
		`p_status`
		`);`

Entrée	p_status	pointeur sur la zone où placer la structure de données WTB_Status.

5.6.4.16.2 Type_Node_Status

Attribut	Type	Signification
node_report	BITSET8	déclaration C correspondant à Node_Report (voir 5.5.2.2)
user_report	BITSET8	déclaration C correspondant à User_Report (voir 5.5.2.3)

5.6.4.16.3 Type_WTBStatus

Attribut	Type	Signification
wtb_hardware_id	UNSIGNED8	identificateur du matériel
wtb_software_id	UNSIGNED8	identificateur de la version du logiciel de couche de liaison.
hardware_state	ENUM8	0: LS_OK, fonctionnement correct 1: LS_FAIL, défaillance du matériel
link_layer_state	LS_T_STATE	voir définition de ce type
net_inhibit	ENUM8	1: un nœud empêche l'inauguration
node_address	UNSIGNED8	adresse du nœud assignée par l'inauguration
node_orient	UNSIGNED8	orientation du nœud par rapport au maître: 0: L_UNKNOWN 1: L_SAME 2: L_INVERSE
node_strength	UNSIGNED8	force du nœud 0: L_UNDEFINED 1: L_SLAVE 2: L_STRONG 3: L_WEAK
node_descriptor	Type_NodeDescriptor	voir définition de ce type
node_status	Type_Node_Status	voir définition de ce type

5.6.4.17 Service Get WTB Nodes

5.6.4.17.1 Type_NodeList

Une structure de données de Type_NodeList doit contenir les éléments suivants:

Attribut		Type	Signification
nr_nodes		UNSIGNED8	nombre de nœuds dans la composition
bottom_node		UNSIGNED8	adresse du Nœud d'Extrémité dans la Direction_1 vue du maître les deux bits de poids fort de cet octet sont 0.
top_node		UNSIGNED8	adresse du Nœud d'Extrémité dans la Direction_2 vue du maître les deux bits de poids fort de cet octet sont 0
node_status_list		ARRAY [MAX_NODES] OF	liste de Node_Status, commençant par le nœud du bas et se terminant par le nœud du haut, dans l'ordre de positionnement des nœuds et qui comprend:
	node_status	Type_Node_Status	voir définition de ce type

5.6.4.17.2 Procédure ls_t_GetWTBNodes

Action	Lit la liste de Node_Report et User_Report de tous les nœuds de la Topographie.	
Syntaxe	`LS_T_RESULT ls_t_GetWTBNodes ( Type_NodeList * p_nodes );`	
Entrée	p_nodes	pointeur sur la position où la liste de nœuds doit être placée.

5.6.4.18 Service Get Topography

5.6.4.18.1 Procédure ls_t_GetTopography

Action	Permet à l'application de lire la Topographie diffusée avant le début du fonctionnement normal.	
Syntaxe	`LS_T_RESULT ls_t_GetTopography ( Type_Topography * p_topography );`	
Entrée	p_topography	pointeur vers la zone de mémoire où mettre la Topographie.
Résultat		renvoie L_CONFIGURATION_INVALID si la Topographie n'est pas valable.

5.6.4.18.2 Type_Topography

La structure de données Topography doit contenir les éléments suivants:

Attribut	Type	Signification
node_address	UNSIGNED8	les deux bits de poids fort de cet octet sont 0. les 6 bits de poids faible de cet octet sont l'adresse du nœud auquel cette station est reliée.
node_orient	UNSIGNED8	Orientation du nœud par rapport au maître: 0: L_UNKNOWN 1: L_SAME 2: L_INVERSE
topo_counter	UNSIGNED8	les six bits de poids faible copient les six bits du Topo_Counter dans le nœud. Les deux bits de poids fort sont 0.
individual_period	UNSIGNED8	période attribuée à un nœud par le maître, comme la puissance de deux de la Période de Base, en millisecondes.
is_strong	UNSIGNED8	1: bus contrôlé par un maître fort, 0: bus contrôlé par un maître faible
number_of_nodes	UNSIGNED8	nombre de nœuds selon les résultats de l'inauguration.
bottom_address	UNSIGNED8	Adresse du Nœud d'Extrémité dans la Direction_1 vue du maître, les deux bits de poids fort de cet octet sont 0.
top_address	UNSIGNED8	Adresse du Nœud d'Extrémité dans la Direction_2 vue du maître, les deux bits de poids fort de cet octet sont 0.
inauguration_data	Type_Inauguration_Data	voir définition de ce type

5.6.4.19 Service Change Node_Descriptor

5.6.4.19.1 Procédure ls_t_ChgNodeDesc

Action	Fournit un nouveau descripteur à la couche de liaison. L'appel de cette procédure pendant le fonctionnement normal provoque l'interruption du trafic et une nouvelle distribution de topographie.	

Syntaxe	`LS_T_RESULT ls_t_ChgNodeDesc (`
	`Type_NodeDescriptor * node_descriptor`
	`);`	

Entrée	node_descriptor	pointeur vers une structure de données Node_Descriptor

5.6.4.20 Service Change User_Report

5.6.4.20.1 Procédure ls_t_ChgUserReport

Action	Permet à l'application de modifier User_Report.	
Syntaxe	`LS_T_RESULT ls_t_ChgUserReport ( UNSIGNED8 set_mask UNSIGNED8 clear_mask );`	
Entrée	set_mask	met les bits définis dans le masque à '1' dans le User_Report.
	clear_mask	met les bits mis à '1' dans le masque à '0' dans le User_Report.

```
LS_T_RESULT          ls_t_ChgUserReport
                     (
    UNSIGNED8        set_mask
    UNSIGNED8        clear_mask
                     );
```

5.6.4.21 Service Change Inauguration_Data

5.6.4.21.1 Procédure ls_t_ChgInauguration_Data

Action	Permet à l'application de modifier les données d'inauguration du nœud.	
Syntaxe	`LS_T_RESULT ls_t_ChgInauguration_Data ( UNSIGNED8 inaug_data_size void* p_inauguration );`	
Entrée	inaug_data_size	taille en octets des données d'inauguration (≤ 124)
	p_inauguration	données d'inauguration définies par l'utilisateur

```
LS_T_RESULT          ls_t_ChgInauguration_Data
                     (
    UNSIGNED8        inaug_data_size
    void*            p_inauguration
                     );
```

5.6.4.22 Service Get Statistics

5.6.4.22.1 Procédure ls_t_GetStatistics

Action	Fournit des statistiques sur l'utilisation et les erreurs.	
Syntaxe	`LS_T_RESULT ls_t_GetStatistics ( Type_LLStatisticData * p_statistic_data );`	
Entrée	p_statistic_data	pointeur vers la structure de données statistiques (voir 5.6.4.22.3)

```
LS_T_RESULT               ls_t_GetStatistics
                          (
Type_LLStatisticData *    p_statistic_data
                          );
```

5.6.4.22.2 Type_LineStatus

Une structure de données du Type_LineStatus doit contenir les éléments suivants:

Attribut	Type	Signification
transmitted_count	UNSIGNED32	Nombre de trames émises par ce nœud
received_count	UNSIGNED32	Nombre de trames reçues sans erreurs par le nœud
errors_count	UNSIGNED16	Nombre de trames erronées reçues
timeouts_count	UNSIGNED16	nombre de temporisations écoulées lorsqu'une réponse est attendue

NOTE Ces compteurs reviennent à 0 après avoir atteint la valeur maximale, leur valeur initiale n'est pas spécifiée.

5.6.4.22.3 Type_LLStatisticData

Une structure de données du Type_LLStatisticData doit contenir les éléments suivants:

Attribut	Type	Signification
basic_period_count	UNSIGNED32	incrémenté de 1 pour chaque Période de Base
inauguration_count	UNSIGNED16	incrémenté de 1 pour chaque nouvelle inauguration
topography_count	UNSIGNED16	incrémenté de 1 pour chaque nouvelle topographie
transmitted_md_count	UNSIGNED32	incrémentéé de 1 pour chaque Message_Data_Response envoyée
received_md_count	UNSIGNED32	incrémenté de 1 pour chaque Message_Data_Response reçue
line_status_a1	Type_LineStatus	voir ce type
line_status_a2	Type_LineStatus	voir ce type
line_status_b1	Type_LineStatus	voir ce type
line_status_b2	Type_LineStatus	voir ce type
line_switch_count	UNSIGNED32	incrémenté de 1 pour chaque commutation vers la ligne redondante

NOTE Ces compteurs reviennent à 0 après avoir atteint la valeur maximale, leur valeur initiale n'est pas spécifiée.

5.6.4.23 Service Get Inauguration_Data

Action	Renvoie un pointeur vers les données d'inauguration	

Syntaxe	`LS_T_RESULT`
	`void * *`
		`ls_t_GetInaugData (`
		`p_inaug_data_list`
		`);`

Sortie	p_inaug_data_list	pointeur vers les données d'inauguration de tous les nœuds nommés.

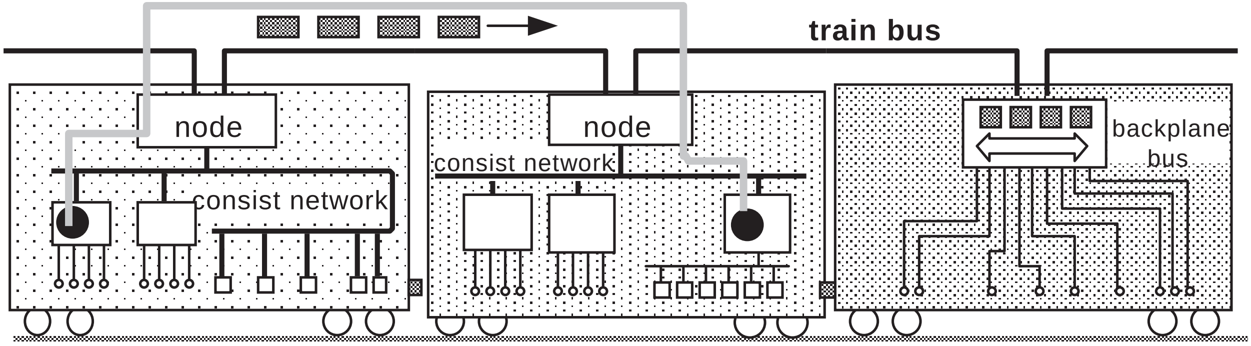
6 Protocoles en Temps Réel

Le présent article s'applique à un réseau TCN qui utilise le WTB et/ou le MVB et/ou tout autre bus répondant aux mêmes principes de fonctionnement.

6.1 Généralités

6.1.1 Teneur du présent article

Le présent article spécifie un élément du Réseau Embarqué de Train, les Protocoles en Temps Réel, qui assure la communication entre les applications à l'intérieur et entre les rames (voir la Figure 107).



Légende

Anglais	Français
node	nœud
train bus	bus de train
consist network	réseau de rame
backplane bus	bus de fond de panier
vehicle bus	bus de véhicule

Figure 107 – Structure du Réseau Embarqué de Train

Le présent article définit les deux services principaux de communication pour l'application:

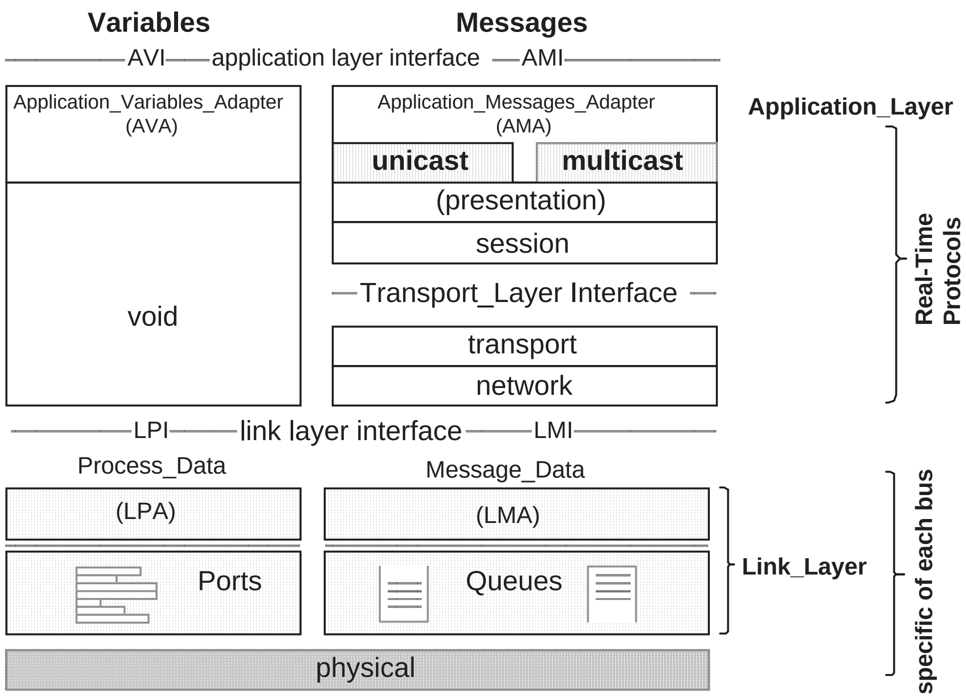
a) Variables: transfert de données courtes et à délai de livraison déterministe, comprenant:
   - l'Interface de la Couche de Liaison pour Process_Data (LPI),
   - l'Interface de la Couche Application pour les Variables (AVI);

b) Messages: transfert de données pouvant être longues, mais non fréquentes, divisées en petits paquets si nécessaire et transmises à la demande, comprenant:
   - l'Interface de la Couche de Liaison pour Message_Data (LMI),
   - la Couche Réseau utilisée pour l'acheminement des paquets à l'intérieur du réseau,
   - la Couche Transport qui assure le contrôle de flux et la reprise sur erreur:
     – pour le point à point, ou
     – les messages publipostés (en option),
   - la Couche Session qui associe Call_Message et Reply_Message,
   - l'Interface de la Couche Application pour les Messages (AMI).

Le présent article définit également la présentation des données (pour les Variables et les Messages).

6.1.2 Structure du présent article

La structure du présent article se base sur le modèle de communication OSI (voir la Figure 108).



Légende

Anglais	Français
variables	variables
messages	messages
application layer interface	interface de couche application
real time protocols	protocoles en temps réel
unicast	point à point
multicast	distribué
transport	transport
network	réseau
void	vide
link layer interface	interface de couche de liaison
ports	ports
queues	files d'attente
specific of each bus	spécifique à chaque bus
physical	physique

Figure 108 – Organisation en couches des Protocoles en Temps Réel

Paragraphe 6.1 - Généralités

Exigences normatives et définitions

Paragraphe 6.2 – Variables – Services et Protocoles

Objets Process_Variable

Interface Dataset

Interface Application pour les accès individuels, par jeu et par grappe

Paragraphe 6.3 – Services de messages et protocoles

Architecture

Interface Couche de Liaison

Couche réseau

Couche transport

Couche session

Interface Application

Paragraphe 6.4 – Présentation et codage des données émises et stockées

6.2 Variables – Services et protocoles

6.2.1 Généralités

Les services et les protocoles sont répartis en interfaces inférieure et supérieure:

a) l'interface de la couche de liaison (inférieure) qui spécifie les services attendus du bus; et

b) l'interface de la couche application qui spécifie les services offerts à l'application.

6.2.2 Interface de la couche de liaison pour Process_Data

6.2.2.1 Objet

L'Interface de liaison de Données de Processus (LPI = Link_Process_Data_Interface) définit les services Process_Data fournis par un bus aux protocoles supérieurs.

La LPI détermine l'initialisation des ports, l'insertion et l'effacement de Datasets complets dans les ports et les primitives de synchronisation associées à l'émission des Datasets complets.

En général, une Application n'a pas d'accès direct à la LPI, sauf pour la synchronisation (pour être informée de la réception ou de l'émission des Datasets).

La communication sous-jacente n'est pas spécifiée par cette interface. La transmission entre les ports, y compris la stratégie d'interrogation du maître du bus, est réalisée par la couche de liaison et la couche physique.

NOTE Les Process_Variables individuelles ne sont pas visibles au niveau LPI.

6.2.2.2 Datasets

6.2.2.2.1 Ports et Traffic_Store

La couche de liaison doit fournir un certain nombre de ports pour communiquer les Process_Data.

Un port est une structure de mémoire partagée à laquelle l'application et le réseau peuvent accéder simultanément.

Un port est une structure de données dépourvue de file d'attente, c'est-à-dire que son contenu est écrasé par une nouvelle valeur écrite et n'est pas affecté par une opération de lecture.

Tant la couche de liaison que l'application doivent pouvoir accéder à un port de manière cohérente, c'est-à-dire écrire ou lire toutes ses données en une seule opération.

Les ports appartenant à la même couche de liaison appartiennent au même Traffic_Store.

A l'intérieur d'un Traffic_Store, un port doit être identifié par sa Port_Address.

A l'intérieur d'un dispositif, un Traffic_Store doit être identifié par sa Traffic_Store_Id.

6.2.2.2.2 Cohérence des Datasets

Chaque Port doit contenir exactement un Dataset.

Un Dataset ne doit être produit que par une seule application éditrice.

Il ne doit y avoir qu'un seul port émetteur sur le bus pour une Port_Address donnée, mais il peut exister un nombre indéterminé de ports destinataires.

Les couches de liaison des différents dispositifs doivent transmettre le contenu d'un port émetteur aux ports destinataires abonnés à la même Port_Address et assurer la cohérence des Datasets transmis.

NOTE Le bus n'est pas supposé garantir que des Datasets différents puissent être transmis ou récupérés comme un ensemble cohérent.

6.2.2.2.3 Traitement d'erreurs

Les champs non définis à l'intérieur d'un Dataset doivent être écrasés par des 1.

Si la couche de liaison ne peut garantir la cohérence d'un Dataset (si elle détecte une erreur de transmission ou que l'application éditrice ne peut fournir les données correctes ou à temps, par exemple), elle doit écraser le port tout entier par des 0.

NOTE Dans la mesure où l'écrasement de la valeur d'une Process_Variable par des 0 ou des 1 peut produire une valeur correcte, une Check_Variable du même Dataset sert d'indicateur de validité, là où un problème surviendrait.

6.2.2.2.4 Contrôle de rafraîchissement

Chaque port destinataire, et donc chaque Dataset, doit disposer d'un Freshness_Timer qui indique le temps écoulé depuis la dernière écriture d'une nouvelle valeur par le bus sur ce port.

Ce Freshness_Timer doit être récupéré en une seule opération avec le contenu du Dataset.

La résolution de Freshness_Timer doit être de 16 ms au maximum.

Il doit avoir une durée d'au moins 4 s et s'arrêter à la fin de cette durée.

NOTE 1 Le Freshness_Timer ne tient pas compte du temps écoulé depuis que l'application éditrice a introduit la Process_Variable dans le port. La supervision du temps à la source concerne l'application qui peut être traitée par des Check_Variables.

NOTE 2 Le Freshness_Timer est indépendant du forçage éventuel des Variables.

6.2.2.2.5 Dataset de synchronisation

La diffusion de certains Datasets peut être utilisée pour synchroniser des applications.

6.2.2.2.6 Interrogation des Datasets

Les procédures d'interrogation des Datasets font partie de la couche de liaison du réseau de rame et du bus de train respectivement. Elles ne sont pas décrites dans la présente norme.

6.2.2.2.7 Indicatif des Dataset (DS_Name)

6.2.2.2.7.1 Dataset, port et Adresse Logique

A l'intérieur d'un dispositif, un Dataset doit être identifié par sa Traffic_Store et par la Port_Address du Traffic_Store dans lequel il est sauvegardé.

Lorsqu'il est transmis sur un bus, un Dataset doit être identifié par la Logical_Address de sa trame Process_Data sur ce bus.

La Logical_Address de la trame Process_Data doit être identique à la Port_Address du Traffic_Store dans lequel le Dataset est sauvegardé.

6.2.2.2.7.2 Format du DS_Name

Un Dataset à l'intérieur d'un dispositif doit être identifié par son DS_Name.

Définition	Type de Dataset

Syntaxe	`typedef struct /* big-endian representation */ {`
	`unsigned traffic_store_id:4, /* DS_NAME first part */`
	`unsigned port_address :12, /* DS_NAME second part */`
	`} DS_NAME;`

Ce DS_Name peut être représenté comme un mot de 16 bits, comme suit:

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

6.2.2.2.7.3 Indicatif du Traffic_Store

Traffic_Store_Id doit sélectionner l'un des Traffic_Stores d'un dispositif.

Le nombre maximal de Traffic_Stores pris en charge doit être de 16.

NOTE 1 L'Indicatif du Traffic_Store ne précise pas le type de bus (MVB, WTB ou autre) auquel il accède, mais cela peut être utile pour simplifier la mise en œuvre (un Traffic_Store WTB peut toujours être 1, par exemple).

NOTE 2 L'Indicatif du Traffic_Store peut être identique au Bus_Id du bus correspondant. Cependant, un Traffic_Store peut exister sans être relié à un bus (dans le cas de communication interne entre les tâches, par exemple).

6.2.2.2.8 Port_Address

La Port_Address doit identifier un des 4096 ports à l'intérieur du Traffic_Store sélectionné par Traffic_Store_Id.

NOTE Le nombre réel de ports dépend du type de bus connecté.

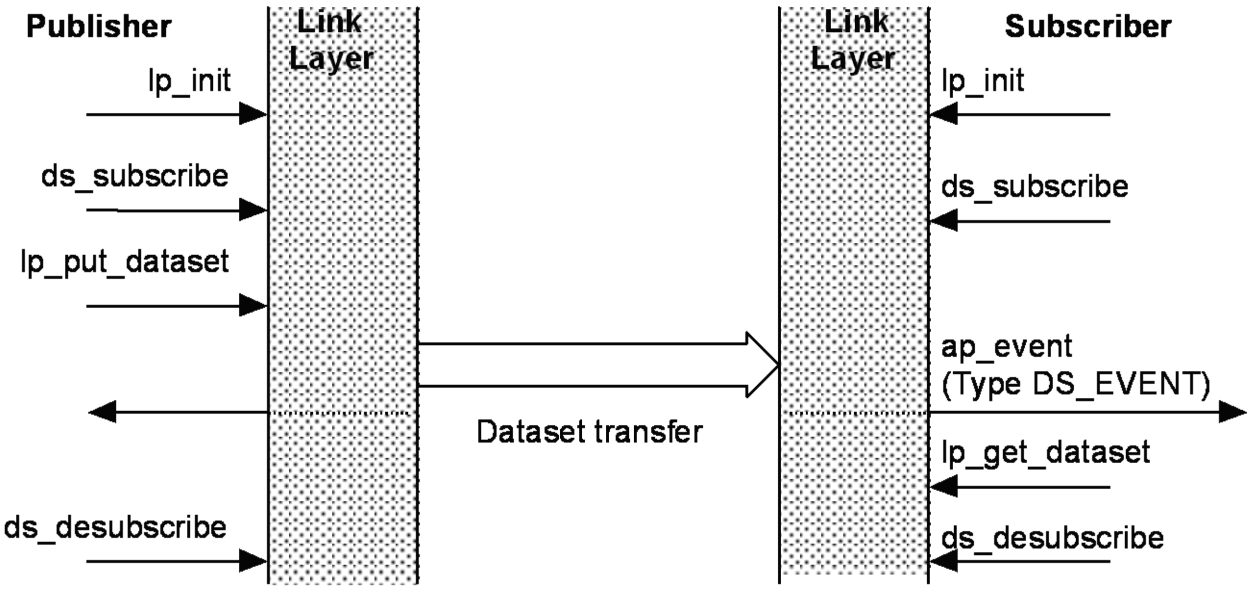
EXEMPLE Sur le MVB, il peut y avoir jusqu'à 4096 ports de 256 bits maximum chacun par dispositif.

6.2.2.3 Primitives de Link_Process_Data_Interface

6.2.2.3.1 Généralités

La Link_Process_Data_Interface (LPI) doit fournir pour l'accès du Dataset les primitives indiquées dans la Figure 109 et détaillées dans le Tableau 18 (voir les paragraphes ci-après).

Les paragraphes qui suivent n'impliquent pas une mise en œuvre particulière. Toute interface qui fournit la même sémantique est autorisée.



Légende

Anglais	Français
publisher	éditeur
link layer	couche de liaison
subscriber	souscripteur
dataset transfer	transfert de Dataset

Figure 109 – Echange de primitives LPI

Tableau 18 – Primitives LPI

Nom	Signification
lp_init	Initialise le Traffic_Store
lp_put_dataset,	Insère un Dataset à transmettre
lp_get_dataset.	Lit un Dataset reçu
ds_subscribe,	Souscrit un Dataset pour la synchronisation
ap_event	Synchronisation en émission ou en réception
ds_desubscribe,	Annule la souscription pour la synchronisation d'un Dataset

NOTE 1 L'application peut accéder directement aux structures Traffic_Store au lieu d'utiliser les primitives pour accélérer l'accès.

NOTE 2 Un processeur de communication peut utiliser ces mêmes primitives pour accéder au Traffic_Store du côté bus.

NOTE 3 Ces primitives ne déclenchent pas immédiatement une communication sur le bus. Elles ne font qu'accéder au Traffic_Store.

6.2.2.3.2 Type LP_RESULT

Définition	Une procédure de la LPI qui renvoie une valeur de type LP_RESULT doit la coder comme suit:

Syntaxe	`typedef enum`
	`{`
	`LP_OK = 0, /* accomplissement correct */`
	`LP_PRT_PASSIVE = 1, /* avertissement: Dataset non activé */`
	`LP_ERROR = 2, /* erreur non spécifiée */`
	`LP_CONFIG = 3, /* erreur de configuration */`
	`LP_MEMORY = 4, /* mémoire insuffisante */`
	`LP_UNKNOWN_TS = 5, /* Traffic_Store inconnu */`
	`LP_RANGE = 6, /* adresse mémoire erronée */`
	`LP_DATA_TYPE = 7 /* type de données non pris en charge */`
	`} LP_RESULT;`

6.2.2.3.3 Procédure lp_init

Définition	Crée un Traffic_Store, établit la liste d'abonnés, crée des ports émetteur et destinataire et les initialise à une valeur prédéfinie.	

Syntaxe	`LP_RESULT lp_init`
	`(`
	`ENUM8 ts_id`
	`void * p_descriptor`
	`);`	

Entrée	ts_id	Traffic_Store_Id (0..15)
	p_descriptor	Structure de données dépendante de la mise en œuvre
Renvoi		toute valeur de LP_RESULT

6.2.2.3.4 Procédure lp_put_dataset

Définition	Copie un Dataset de l'application vers un port de Traffic_Store.	

Syntaxe	`LP_RESULT lp_put_dataset`
	`(`
	`DS_NAME * dataset;`
	`void * p_value`
	`);`	

Entrée	dataset	DS_Name du Dataset à publier
	p_value	pointeur vers une zone mémoire de l 'Application d'où la valeur du Dataset est copiée.
Renvoi		toute valeur de LP_RESULT
Usage	La valeur précédente du Dataset dans le Traffic_Store est écrasée.	

6.2.2.3.5 Procédure lp_get_dataset

Définition	Copie un Dataset et son Freshness_Timer d'un port vers l'Application.	
Syntaxe	`LP_RESULT lp_get_dataset ( DS_NAME * dataset; void * p_value; void * p_fresh );`	
Entrée	dataset	DS_Name du Dataset à recevoir.
Renvoi		toute valeur de LP_RESULT
Sortie	p_value	Pointeur vers une Memory_Address de l'Application où la valeur du Dataset est copiée.
	p_fresh	Pointeur vers une Memory_Address de l'Application où le Freshness_Timer est copié.

```
LP_RESULT          lp_get_dataset
                   (
    DS_NAME *      dataset;
    void *         p_value;
    void *         p_fresh
                   );
```

6.2.2.3.6 Procédure ds_subscribe

Définition	Souscrit un Dataset pour la transmission ou la réception et indique la procédure d'indication appelée si le Dataset spécifié est transmis ou reçu.	
Syntaxe	`LP_RESULT Ds_subscribe ( DS_NAME * dataset; DS_EVENT event_cnf; UNSIGNED16 instance );`	
Entrée	dataset	DS_Name du Dataset à inclure dans la souscription.
	event_cnf	Procédure souscrite
	instance	Numéro de référence de 16 bits qui identifie l'instance d'application à souscrire et qui sera renvoyée dans la procédure ds_event.
Renvoi		toute valeur de LP_RESULT
Usage	1 – Cette procédure peut être appelée plusieurs fois pour différents Datasets et procédures souscrites, le nombre de fois étant limité par l'application. 2 – Un Dataset donné ne peut être souscrit qu'une seule fois.	

```
LP_RESULT          Ds_subscribe
                   (
    DS_NAME *          dataset;
    DS_EVENT           event_cnf;
    UNSIGNED16         instance
                   );
```

6.2.2.3.7 Type DS_EVENT

Définition	Lorsqu'un Dataset a été envoyé ou reçu, la Couche de Liaison doit appeler la procédure souscrite à ce Dataset, qui doit être du type DS_EVENT	

Syntaxe	`typedef void ( * DS_EVENT)`
	`(`
	`UNSIGNED16 instance`
	`);`	

Entrée	instance	Numéro de référence de 16 bits qui identifie l'instance d'application qui a souscrit cet événement.
Usage	1 – Cette procédure a été souscrite précédemment par ds_subscribe. 2 – Le Dataset qui a provoqué cet événement n'est pas identifié, mais le paramètre d'instance peut être utilisé pour l'identifier	

6.2.2.3.8 Procédure ds_desubscribe

Définition	Résilie la souscription à un Dataset	

Syntaxe	`LP_RESULT Ds_desubscribe`
	`(`
	`DS_NAME * dataset;`
	`);`	

Entrée	dataset	DS_Name du Dataset à effacer de la souscription
Renvoi		toute valeur de LP_RESULT

6.2.3 Interface d'application pour Process_Variables

6.2.3.1 Objet

L'interface d'application des variables (AVI, pour Application_Variable_Interface) définit les services de transfert de Variables offerts à l'Application.

Les primitives de cette interface n'accèdent qu'aux ports dans le (ou les) Traffic_Store(s) et ne déclenchent pas de communication sur le bus.

L'introduction d'une variable par une application éditrice dans un port est censée entraîner, dans un temps limité, l'introduction de cette même Variable dans le port correspondant du (des) souscripteur(s).

6.2.3.2 Process_Variables

6.2.3.2.1 Émission et sauvegarde de Process_Variable

Les Process_Variables sont émises comme partie intégrante d'un Dataset.

Toutes les Process_Variables appartenant à un Dataset doivent être émises et sauvegardées comme un ensemble cohérent.

6.2.3.2.2 Contrôle de rafraîchissement

Une Process_Variable doit être récupérée avec le Freshness_Timer associé à son Dataset en une seule opération.

6.2.3.2.3 Synchronisation

Une application peut être synchronisée par l'émission d'un Dataset à travers la LPI.

6.2.3.2.4 Check_Variable

Pour évaluer sa validité, chaque Variable peut être accompagnée d'une autre Variable du même Dataset, appelée la Check_Variable.

Check_Variable et Process_Variable doivent être stockées et récupérées en une seule opération.

La même Check_Variable peut s'appliquer à plusieurs Process_Variables.

La Check_Variable peut se trouver à n'importe quel endroit dans le Dataset et peut recouvrir une Process_Variable.

Si la Check_Variable est utilisée, elle doit avoir le format d'un ANTIVALENT2 et prendre les valeurs suivantes:

a) '00'B: les variables protégées sont erronées ou suspectes;
b) '01'B: les variables protégées sont présumées correctes;
c) '10'B: la valeur des variables protégées a été forcée à une valeur imposée;
d) '11'B: les variables protégées ne sont pas définies.

NOTE 1 Le mot Variable est utilisé quand il n'est pas précisé s'il s'agit d'une Process_Variable ou d'une Check_Variable.

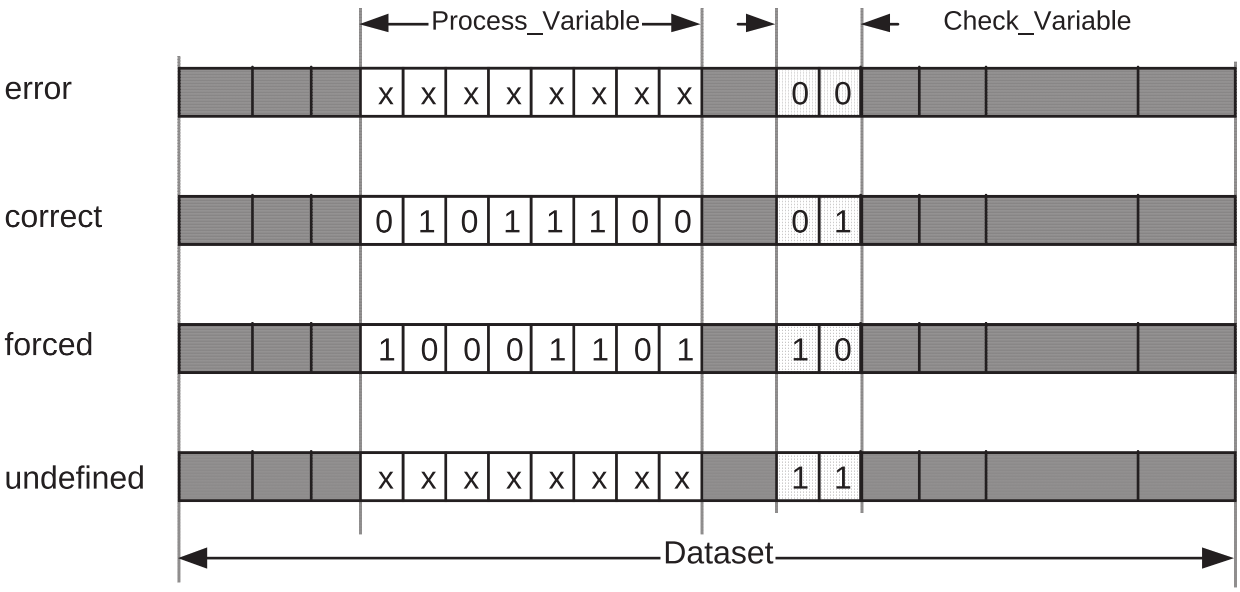
NOTE 2 L'application se charge de l'affectation des Process_Variables et des Check_Variables dans le Dataset.

NOTE 3 Le bus ou l'application éditrice écrasent par des '0' les champs douteux d'un Dataset. Cela permet de positionner les deux bits de Check_Variable à '00'B, permettant ainsi à l'application de détecter les erreurs.

NOTE 4 Il convient d'écraser par des 1 le champ réservé pour une extension future. Cela permet de positionner les deux bits des Check_Variables à '11'B, permettant à des dispositifs futurs de ne pas prendre les 1 pour des données valides quand ils les reçoivent d'un dispositif plus ancien.

NOTE 5 L'application est supposée traiter les deux situations possibles provoquées par des problèmes de communication (tout à '0') ou par les données invalidées (tout à '1').

EXEMPLE La Figure 110 illustre une Process_Variable et sa Check_Variable associée dans le même Dataset.



Légende

Anglais	Français
error	erreur
correct	correct
forced	forcé
undefined	non défini

Figure 110 – Check_Variable

6.2.3.2.5 Indicatif de Process_Variable (PV_Name)

6.2.3.2.5.1 Indicatif de Variable et Dataset

A l'intérieur d'un dispositif, une Process_Variable doit être identifiée par son Dataset (DS_Name) et son décalage en bits à l'intérieur du Dataset (Var_Offset).

Lorsqu'elle est transmise sur un bus, une Process_Variable doit être identifiée par sa Logical_Address et son décalage en bits à l'intérieur du Dataset transmis (Var_Offset).

6.2.3.2.5.2 Format de PV_Name

A l'intérieur d'un dispositif, chaque Process_Variable doit être identifiée par un indicatif unique, appelé PV_Name, qui comprend les éléments suivants:

a) Traffic_Store_Id;
b) Port_Address;
c) Var_Offset;
d) Var_Size;
e) Var_Type;
f) Chk_Offset.

NOTE D'un dispositif à l'autre, le PV_Name de la même Variable du même bus peut être différent dans son Traffic_Store_Id, car l'indicatif du Traffic_Store peut varier.

6.2.3.2.5.3 Indicatif du Traffic_Store

Le Traffic_Store_Id doit identifier l'un des 16 Traffic_Stores à l'intérieur d'un dispositif.

6.2.3.2.5.4 Port_Address

Le Port_Address doit identifier l'un des 4096 ports à l'intérieur de Traffic_Store.

6.2.3.2.5.5 Var_Offset

Si un Dataset contenait un nombre entier non signé, son bit de poids fort aurait un décalage de '0'.

Le Var_Offset doit définir le décalage, exprimé en bits depuis le début du Dataset, du début du champ occupé par la valeur d'une Process_Variable.

Une Process_Variable doit se trouver à un Var_Offset qui est un multiple de sa taille.

NOTE L'alignement est une concession aux compilateurs qui ne peuvent accéder aux structures de données qui dépassent une limite de mot.

6.2.3.2.5.6 Var_Type et Var_Size

Var_Type et Var_Size doivent identifier de manière unique le format de Process_Variable, Var_Type indiquant le type selon les spécifications de 6.4 et Var_Size la taille.

Var_Size et Var_Type doivent être codés conformément au Tableau 19.

Tableau 19 – Codage de Var_Size et Var_Type dans un PV_Name

Var_Size	Var_Type	Type de données
0	0	BOOLEAN1
	1	ANTIVALENT2
	2	BCD4 ou ENUM4
	3	réservé
	4	BITSET8
	5	UNSIGNED8 ou ENUM8
	6	INTEGER8
	7	CHARACTER8 (ARRAY [0..0] OF WORD8)
1	4	BITSET16
	5	UNSIGNED16 ou ENUM16
	6	INTEGER16
	8	BIPOLAR2.16 (±200 %)
	9	UNIPOLAR2.16 (+400 %)
	10	BIPOLAR4.16(±800 %)
2	3	REAL32
	4	BITSET32
	5	UNSIGNED32 ou ENUM32
	6	INTEGER32
3	2	TIMEDATE48
4	4	BITSET64
	5	UNSIGNED64
	6	INTEGER64
n-1	7	ARRAY OF WORD8 (nombre impair d'octets)
	15	ARRAY OF WORD8 (nombre pair d'octets)
	13	ARRAY OF UNSIGNED16 (n = taille du tableau en WORD16)
	14	ARRAY OF INTEGER16
	11	ARRAY OF UNSIGNED32 (n = taille du tableau en WORD16)
	12	ARRAY OF INTEGER32

Var_Size est interprétée différemment selon qu'il s'agit de types structurés ou primitifs:

- pour les types primitifs, Var_Size indique le nombre de mots de 16 bits utilisés;
- pour les types structurés, Var_Size indique le nombre de mots de 16 bits moins 1.

NOTE 1 Dans les types primitifs, Var_Size = 0 représente moins d'un WORD16, Var_Size = 1 représentant un seul WORD16.

NOTE 2 Dans les types structurés, Var_Size = 0 représente un seul WORD16.

NOTE 3 Le facteur 'n' est le nombre de WORD16. La taille maximale d'une variable est donc 128 octets ($64 \times 16 = 1024$ bits), le Var_Size étant '3F'H.

NOTE 4 Les codes non spécifiés dans le Tableau 19 sont réservés.

NOTE 5 Bien que le triplet {Traffic_Store, Port_Address, Var_Offset} identifie de manière unique une Process_Variable, le PV_Name inclut les informations de type et de taille pour convertir rapidement entre les types de données de réseau et d'application.

6.2.3.2.5.7 Chk_Offset

Le Chk_Offset doit définir la position de Check_Variable associée à la Process_Variable par rapport au début du Dataset.

Un Chk_Offset qui correspond à la position la plus à droite à l'intérieur du Dataset (‘0FFF’H) doit être utilisé lorsqu'une Process_Variable n’est associée à aucune Check_Variable.

6.2.3.3 Primitives de Application_Variables_Interface

6.2.3.3.1 Généralités

Les primitives de l'AVI (Application_Variables_Interface) sont divisées en trois groupes:

a) accès individuel;
b) accès par jeu;
c) accès par grappe.

Les paragraphes qui suivent n'impliquent pas une mise en œuvre particulière. Toute interface qui fournit la même sémantique est autorisée.

6.2.3.3.2 Type AP_RESULT

Définition	Une procédure de l’AVI qui renvoie une valeur doit la coder comme suit:

Syntaxe	<pre>Typedef enum
	  {
	  AP_OK = 0, /* accomplissement correct
	  AP_PRT_PASSIVE = 1, /* avertissement: Dataset non activé
	  AP_ERROR = 2, /* erreur générale
	  AP_CONFIG = 3, /* erreur de configuration
	  AP_MEMORY = 4, /* mémoire insuffisante
	  AP_UNKNOWN_TS = 5, /* Traffic_Store inconnu
	  AP_RANGE = 6, /* Adresse mémoire erronée
	  AP_DATA_TYPE = 7 /* type de données non pris en charge
	  } AP_RESULT;</pre>

NOTE Le codage de ces constantes est identique aux constantes LPI portant un nom similaire.

6.2.3.3.3 Paramètres de Configuration

AP_TS_ID_MAX	0..15	nombre maximal de Traffic_Stores pris en charge dans une mise en œuvre = AP_TS_ID_MAX + 1

6.2.3.3.4 Initialisation des Variables

Les Process_Variables sont initialisées par le mécanisme d'initialisation du Dataset dépendant d’une application pour tous les Datasets.

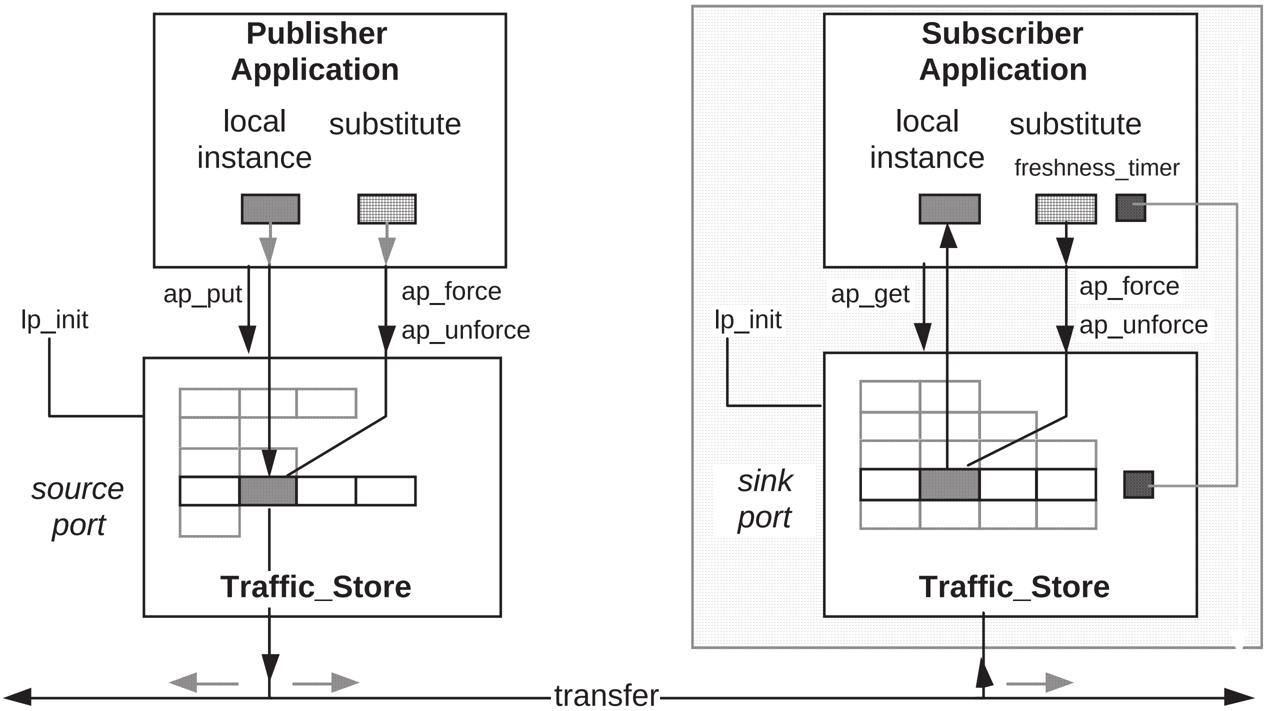
NOTE La couche de liaison initialise par défaut tous les Datasets à ‘0’.

6.2.3.3.5 Primitives d'accès individuel

L’AVI doit assurer l'accès à des variables individuelles par les primitives suivantes, illustrées à la Figure 111 et définies dans les paragraphes suivants:

a) ap_put_variable,

a) ap_put_variable,
b) ap_get_variable,
c) ap_force_variable,
d) ap_unforce_variable,
e) ap_unforce_all.



Légende

Anglais	Français
publisher application	application éditeur
subscriber application	application souscripteur
local instance	instance locale
substitute	substitution
source port	port émetteur
sink port	port destinataire
transfer	transfert

Figure 111 – Accès individuel

6.2.3.3.5.1 Type PV_NAME

Pour l'accès individuel, Var_Offset et Chk_Offset doivent être composés chacune de deux champs, Var_Octet_Offset et Var_Bit_Number.

Var_Octet_Offset est le décalage compté en octets depuis le début du Dataset, le décalage du premier octet transmis ou stocké étant 0.

Pour les variables constituées de plusieurs octets, Var_Bit_Number est toujours 0. Dans le cas des variables inférieures à un octet, Var_Bit_Number représente le nombre de bits dont la variable doit être décalée vers la droite, de manière à justifier la variable d'un octet vers la droite. Var_Bit_Number n'est pas identique au décalage des bits dans l'octet.

<table>
<tr><td>Définition</td><td>Type d'une Process_Variable individuelle</td></tr>
<tr><td>Syntaxe</td><td><pre>typedef struct   /* bit de poids fort en premier
    */
  {
  unsigned traffic_store_id:4, /* DS_NAME première partie
    */
  unsigned port_address :12, /* DS_NAME deuxième partie */
  unsigned var_size   :6, /* voir Tableau 19 */
  unsigned var_octet_offset:7, /* premier octet a offset 0
    */
  unsigned var_bit_number:3, /* compté depuis la droite */
  unsigned var_type   :6, /* voir Tableau 19 */
  unsigned chk_octet_offset:7, /* premier octet a offset 0
    */
  unsigned chk_bit_number:3, /* compté depuis la droite */
  } PV_NAME;</pre></td></tr>
</table>

Le PV_Name peut être codé comme suit:

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="4">Traffic_store_id</td><td colspan="12">port_address</td></tr>
<tr><td colspan="6">var_size</td><td colspan="7">var_octet_offset</td><td colspan="3">var_bit_number</td></tr>
<tr><td colspan="6">var_type</td><td colspan="7">chk_octet_offset</td><td colspan="3">chk_bit_number</td></tr>
</table>

NOTE La définition de Var_Bit_Number permet d'accélérer l'accès individuel, et en particulier d'éviter d'ajouter ou de retrancher huit pour tirer parti des instructions de décalage dans les processeurs. Cette décomposition est uniquement possible si tous les types de données sont alignés, par exemple, un ANTIVALENT2 ne peut pas avoir de décalage impair.

EXEMPLE Le contenu de mémoire suivant représente un PV_Name:

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

Ce PV_Name identifie une Process_Variable qui:

- se trouve dans le Traffic_Store 3, à la Port_Address 'BA'H (= 442), avec un décalage '0F8'H ($31 \times 8 = 248$) bits;
- est du type INTEGER8 (var_size = $0 \times$ WORD16, var_type = 6);
- sa Check_Variable à 2 bits associée est située au décalage d'octet 0, bit numéro 4, c'est-à-dire se trouve dans les troisième et quatrième bits du dataset (décalage de bit 2 et 3). Si ce numéro est impair, aucune variable de contrôle n'est associée.

6.2.3.3.5.2 Procédure ap_put_variable

Définition	Copie une Process_Variable individuelle et sa Check_Variable associée de l'espace Memory_Address de l'application vers un Traffic_Store.	
Syntaxe	AP_RESULT PV_NAME* void * void *	ap_put_variable (ts_variable, p_value, p_check);
Entrée	ts_variable	PV_Name de la Process_Variable
	p_value	pointeur vers une position mémoire de l'application à partir de laquelle la valeur publiée est copiée.
	p_check	pointeur vers une position mémoire de l'application à partir de laquelle la Check_Variable associée est copiée.
Renvoi		tout AP_RESULT
Usage	1 – Si la Process_Variable a été forcée, ap_put_variable n'a aucun effet. 2 – La valeur précédente de la Process_Variable est écrasée. 3 – Les autres données du même Dataset ne sont pas concernées, mais leur cohérence n'est pas garantie.	

6.2.3.3.5.3 Procédure ap_get_variable

Définition	Copie une Process_Variable et ses Freshness_Timer et Check_Variable associés d'un Traffic_Store vers l'application.	
Syntaxe	AP_RESULT PV_NAME* void * void * void *	ap_get_variable (ts_variable, p_value, p_check, p_fresh);
Entrée	ts_variable	PV_Name de la Process_Variable
	p_value	pointeur vers une position mémoire de l'application où est placée la valeur reçue.
	p_check	pointeur vers une position mémoire de l'application où est placée la Check_Variable associée.
	p_fresh	pointeur vers une position mémoire de l'application où est placé le Freshness_Timer associé.
Renvoi		tout AP_RESULT
Usage	1 – Cette primitive peut être utilisée avec un port émetteur ou destinataire, pour permettre aux souscripteurs d'accéder au même dispositif que l'éditeur. 2 – Si la Process_Variable a été forcée, la valeur forcée est récupérée.	

6.2.3.3.5.4 Procédure ap_force_variable

Définition	Force une Process_Variable individuelle d'un port à une valeur spécifiée; met la Check_Variable associée à la valeur '10'B.	
Syntaxe	`AP_RESULT ap_force_variable ( PV_NAME * ts_variable, void * p_value );`	
Entrée	ts_variable	PV_Name de la Process_Variable
	p_value	pointeur vers une position mémoire de l'application d'où la valeur forcée est copiée.
Renvoi		tout AP_RESULT
Usage	Le type de la valeur substituée est supposé être compatible avec celui du PV_NAME.	

6.2.3.3.5.5 Procédure ap_unforce_variable

Définition	Termine le forçage d'une Variable et restaure son accès normal au bus; ne modifie pas la Check_Variable correspondante.	
Syntaxe	`AP_RESULT ap_unforce_variable ( PV_NAME * ts_variable );`	
Entrée	ts_variable	PV_Name de la Process_Variable
Renvoi		tout AP_RESULT

6.2.3.3.5.6 Procédure ap_unforce_all

Définition	Termine la substitution de toutes les Variables d'un Traffic_Store, ne modifie pas les Check_Variables correspondantes.	
Syntaxe	`AP_RESULT ap_unforce_all ( ENUM8 ts_id );`	
Entrée	ts_id	Traffic_Store_Id (0..15)
Renvoi		tout AP_RESULT

6.2.3.3.6 Procédures d'accès par jeu

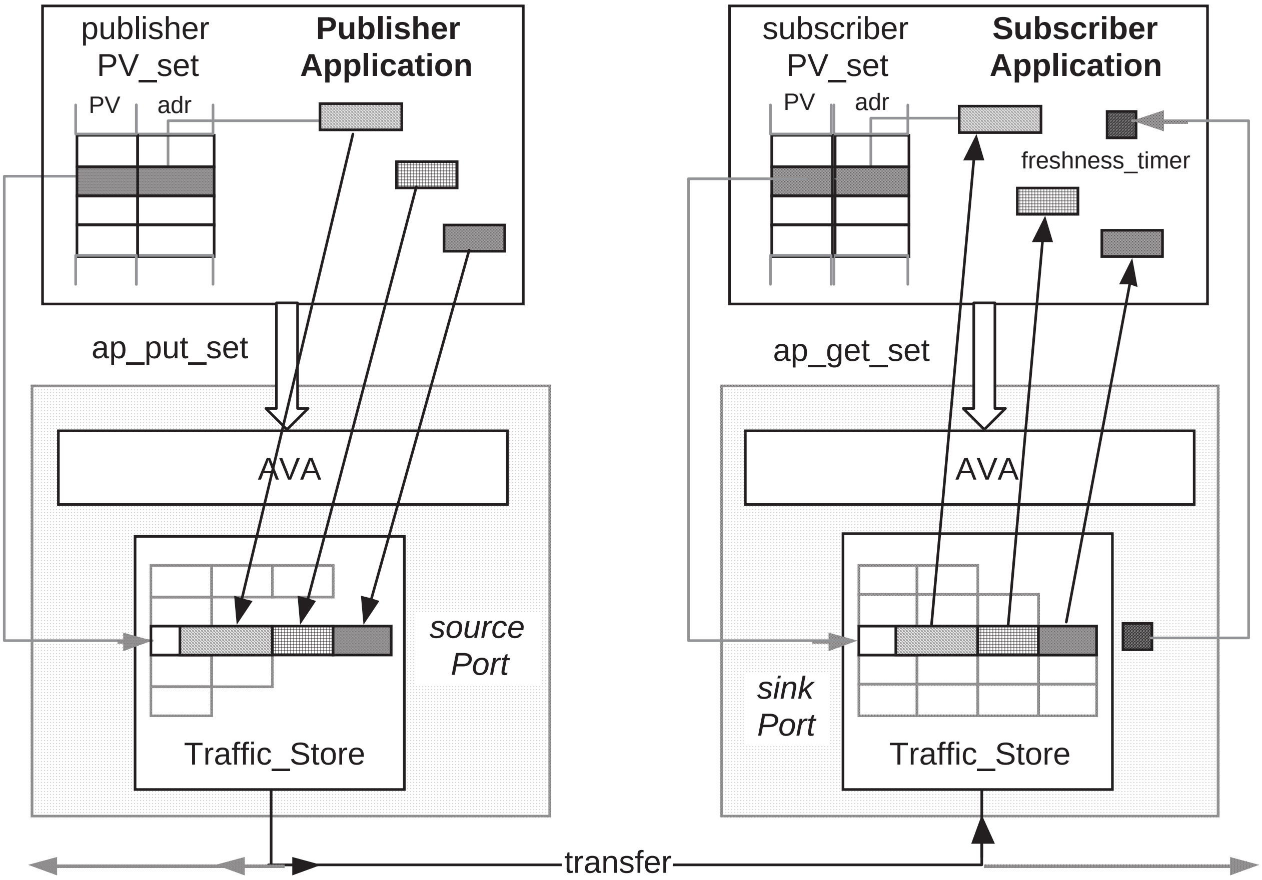
6.2.3.3.6.1 Mode d'accès par jeu

Un jeu est un groupe de Variables (Process_Variables et Check_Variables) appartenant au même Dataset et traité comme un ensemble afin de conserver les informations de cohérence et de rafraîchissement.

L'AVI doit assurer l'accès au jeu par les primitives suivantes, illustrées à la Figure 112 et spécifiées dans les paragraphes suivants:

a) ap_put_set,

b) ap_get_set.



Légende

Anglais	Français
publisher application	application éditeur
subscriber application	application souscripteur
source port	port émetteur
sink port	port destinataire
transfer	transfert

Figure 112 – Accès par jeu

NOTE L'accès par jeu est cohérent, c'est-à-dire que toutes les variables ou aucune sont copiées en une seule opération.

6.2.3.3.6.2 Type PV_SET

Définition	Un PV_Set identifie un jeu de Variables appartenant au même Dataset, et comprend pour chaque variable la Memory_Address sur laquelle (ou de laquelle) il est copié, y compris le Freshness_Timer de l'ensemble du Dataset.	

Syntaxe	`typedef struct PV_LIST {`
	`void* p_variable`
	`UNSIGNED8 derived_type;`
	`UNSIGNED8 array_count;`
	`UNSIGNED8 octet_offset;`
	`UNSIGNED8 bit_number;`
	`};`	

	`typedef PV_SET {`
	`struct PV_LIST* p_pv_list;`
	`UNSIGNED16 c_pv_list;`
	`UNSIGNED16 * p_freshtime;`
	`DS_NAME dataset;`
	`};`	

Éléments	p_variable	Memory_Address de la Variable
	derived_type	type de données général déduit de Var_Type et de Var_Size, dépendant de la mise en œuvre.
	array_count	nombre d'éléments dans le tableau.
	octet_offset	décalage d'une Variable en nombre d'octets
	bit_number	nombre de bits d'une Process_Variable inférieure à un octet ou Check_Variable (voir définition du PV_Name)
	p_pv_list	pointeur vers la PV_List
	c_pv_list	nombre de Variables dans la PV_List
	p_freshtime	Memory_Address de Freshness_Timer. (non utilisé pour ap_put_set)
	dataset	DS_Name (valable pour tout le jeu)
Usage	1 – Les Process_Variables et Check_Variables sont traitées de la même façon, puisque toutes les variables d'un PV_Set sont cohérentes. Il n'y a donc aucune distinction entre Var_Offset et Chk_Offset. 2 – Le Var_Offset (ou Chk_Offset) est divisé en décalage d'octet et en décalage de bit pour accélérer le traitement. Pour la même raison, le type et la taille occupent chacun un octet au lieu des six bits utilisés pour le type de PV_NAME. 3 – Ce même format est utilisé pour le jeu de souscripteur et le jeu d'éditeur, bien que le compteur de rafraîchissement ne soit pas utilisé dans un jeu d'éditeur. 4 – Pour améliorer l'efficacité d'accès, le PV_Set peut également comporter des références directes aux structures de données internes d'un Traffic_Store.	

NOTE Pour des raisons d'efficacité, le PV_SET n'inclut pas le PV_Name complet de chaque Variable. En particulier, la Check_Variable apparaît comme un ANTIVALENT2 normal, car la même Check_Variable peut protéger plusieurs Variables.

6.2.3.3.6.3 Procédure ap_put_set

Définition	Copie une liste de Variables appartenant au même jeu à partir de l'espace Memory_Address de l'application vers le port, en une seule opération.	
Syntaxe	`AP_RESULT ap_put_set ( PV_SET * pv_set );`	
Entrée	pv_set	pointeur vers une PV_List
Renvoi		tout AP_RESULT

6.2.3.3.6.4 Procédure ap_get_set

Définition	Copie une liste de Variables appartenant au même Jeu à partir du port vers l'espace Memory_Address de l'application, en une seule opération.	
Syntaxe	`AP_RESULT ap_get_set ( PV_SET * pv_set );`	
Entrée	pv_set	pointeur vers une PV_List
Renvoi		tout AP_RESULT

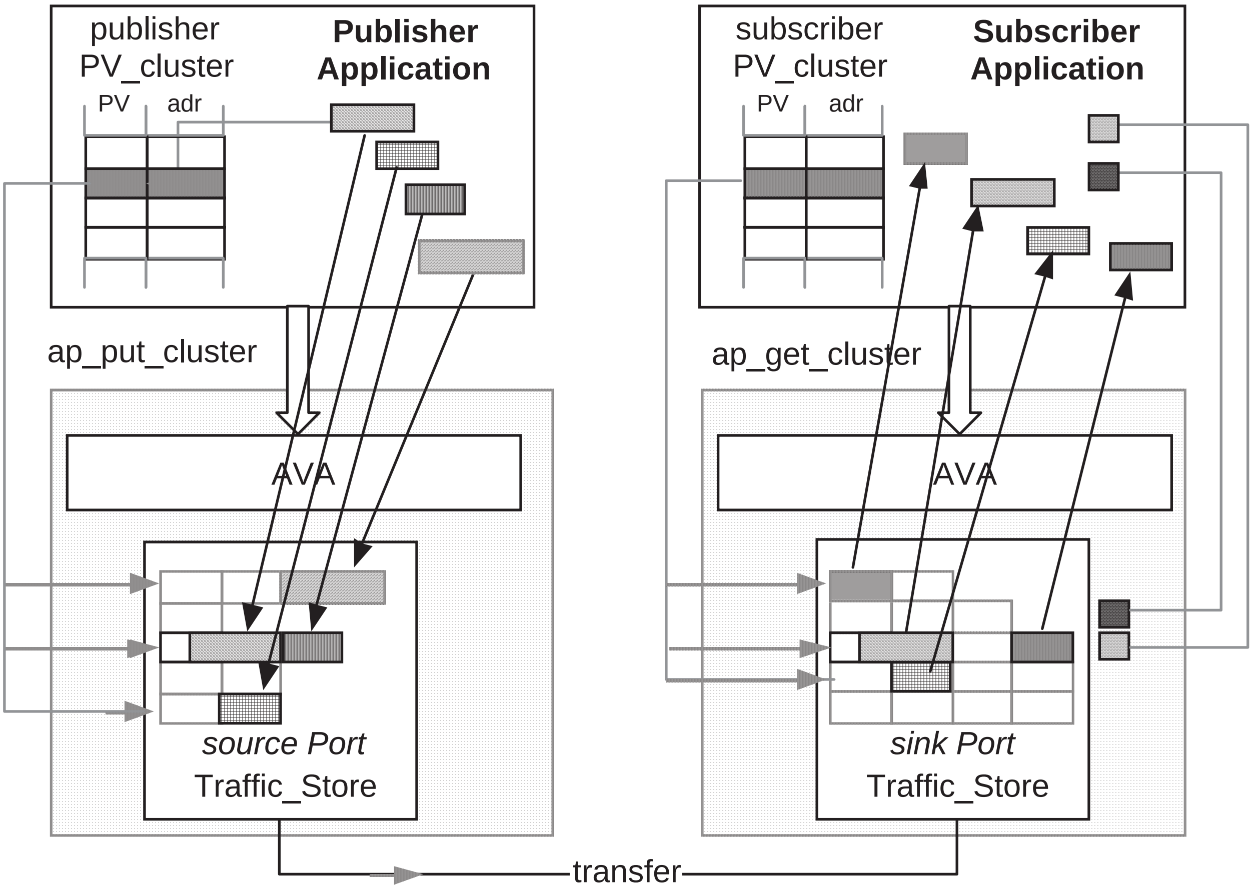
6.2.3.3.7 Procédure d'accès par grappe

6.2.3.3.7.1 Mode d'accès par grappe

Les Grappes sont des groupes de Variables réparties sur plusieurs Datasets et sur plusieurs Traffic_Stores.

L'AVI doit assurer l'accès par grappe par les primitives suivantes, illustrées à la Figure 113 et définies dans les paragraphes suivants:

- ap_put_cluster,
- ap_get_cluster.



Légende

Anglais	Français
publisher application	application éditeur
subscriber application	application souscripteur
source port	port émetteur
sink port	port destinataire
transfer	transfert

Figure 113 – Accès par grappe

NOTE 1 L'accès par grappe ne garantit pas la cohérence de l'ensemble de la grappe, mais seulement celle des PV_Sets individuels à l'intérieur de la grappe.

NOTE 2 Aucune cohérence n'est garantie entre les Variables d'un même Dataset qui apparaissent dans différents PV_Sets.

6.2.3.3.7.2 Type PV_CLUSTER

Un PV_Cluster identifie un groupe de PV_Sets, ordonnés par Traffic_Store.

Définition	Type d'un PV_Cluster	

Syntaxe	`typedef struct`
	
	 `UNSIGNED8`
	 `UNSIGNED8`
	 `struct PV_SET *`
		`PV_CLUSTER`
		 `{`
		 `ts_id;`
		 `c_pv_set;`
		 `p_pv_set [c_pv_set]`
		 `}`

Éléments	ts_id	Traffic_Store_Id (0..15) d'un Traffic_Store
	c_pv_set	nombre de PV_Sets dans la grappe
	p_pv_set	Pointeurs ARRAY [0..c_pv_set -1] OF vers PV_SET
Usage	1 – Il existe une Liste de Grappe pour chaque Traffic_Store. 2 – Le même format est utilisé pour la Liste de Grappe de souscripteurs et la Liste de Grappe d'éditeurs, bien que le compteur de rafraîchissement ne soit pas utilisé dans une Liste de Grappe d'éditeurs. 3 – Pour améliorer l'efficacité d'accès, le PV_Cluster peut également contenir des références directes aux structures de données internes d'un Traffic_Store.	

6.2.3.3.7.3 Procédure ap_put_cluster

Définition	Copie une grappe de Variables de l'application vers le Traffic_Store. Les variables appartenant au même PV_Set sont copiées de manière cohérente	

Syntaxe	`AP_RESULT`
	
	 `    PV_CLUSTER*`
		`ap_put_cluster`
		 `(`
		 `pv_cluster`
		 `);`

Entrée	pv_cluster	pointeur vers une Liste de Grappe d'éditeurs
Renvoi		tout AP_RESULT

6.2.3.3.7.4 Procédure ap_get_cluster

Définition	Copie une grappe de Process_Variables du (des) Traffic_Store(s) vers les instances de souscripteurs locaux. Les variables appartenant au même PV_Set sont copiées de manière cohérente.	

Syntaxe	`AP_RESULT`
	
	 `    PV_CLUSTER*`
		`ap_get_cluster`
		 `(`
		 `pv_cluster`
		 `);`

Entrée	pv_cluster	pointeur vers une Liste de Grappe de souscripteurs
Renvoi		tout AP_RESULT

6.3 Services et Protocoles de Messagerie

6.3.1 Généralités

Les services et protocoles de Messagerie sont séparés en interface inférieure et interface supérieure:

a) l'interface de couche de liaison (inférieure) qui précise les services attendus du bus; et

b) les services de couche application (supérieure) qui sont offerts à l'application.

6.3.2 Station de référence

Une station Réseau Embarqué de Train qui fournit les Services de Messagerie doit contenir les éléments suivants:

a) au moins une connexion bus accessible par sa LPI et qui met en œuvre un processus de couche de liaison;

b) une machine protocole, appelée Messager, qui met en œuvre le réseau, le transport, la session, la présentation et la couche application;

c) un ou plusieurs processus d'application, dont l'un est l'Agent de gestion de réseau.

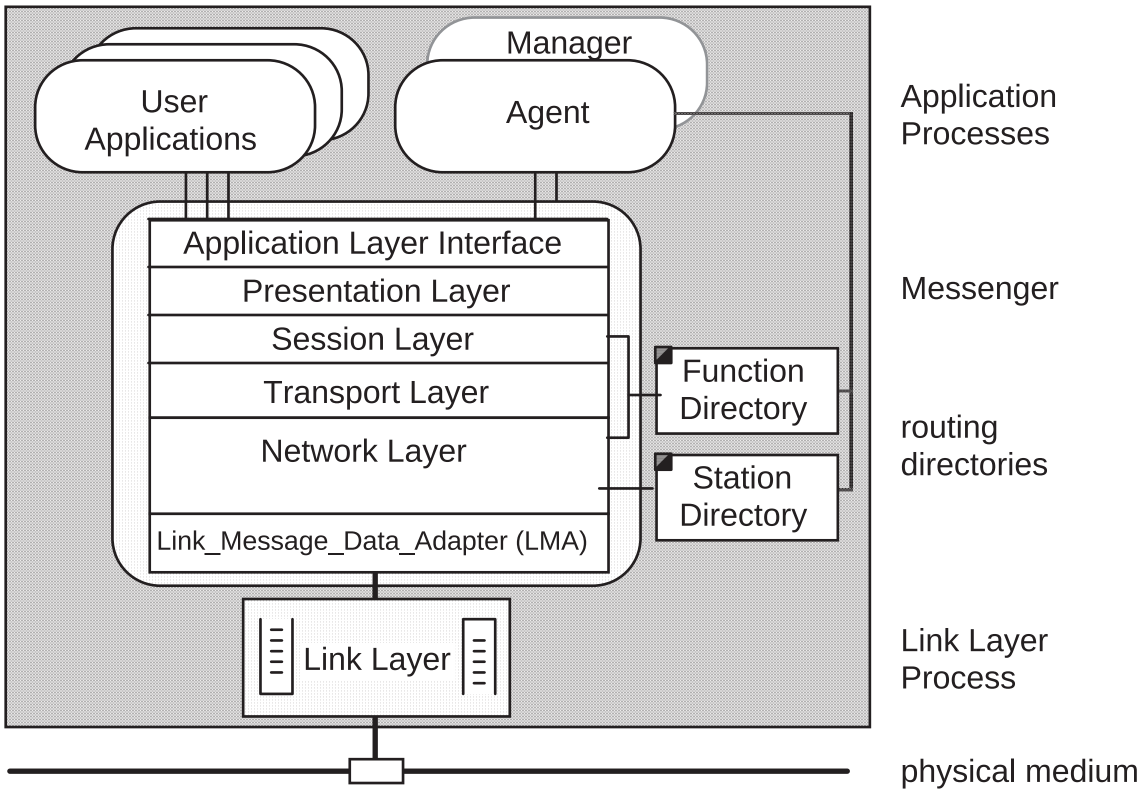
Une station de Gestion doit fournir un Processus de gestion d'application.

NOTE Le jeu minimal de services qu'un Agent est censé fournir est spécifié dans l'Article 8 (Train_Network_Management).

6.3.2.1 Station terminale

Les stations reliées à un seul bus ou les stations terminales ne doivent comporter qu'une seule couche de liaison.

EXEMPLE La Figure 114 présente la structure d'une station terminale de référence.



Légende

Anglais	Français
User applications	applications utilisateur
manager	gestionnaire
agent	agent
application processes	processus d'application
application layer interface	interface de la couche application
presentation layer	couche présentation
session layer	couche session
transport layer	couche transport
network layer	couche réseau
messenger	messager
routing directories	répertoires d'acheminement
function directory	répertoire de fonctions
station directory	répertoire de stations
link layer process	processus de la couche de liaison
link layer	couche de liaison
physical medium	support physique

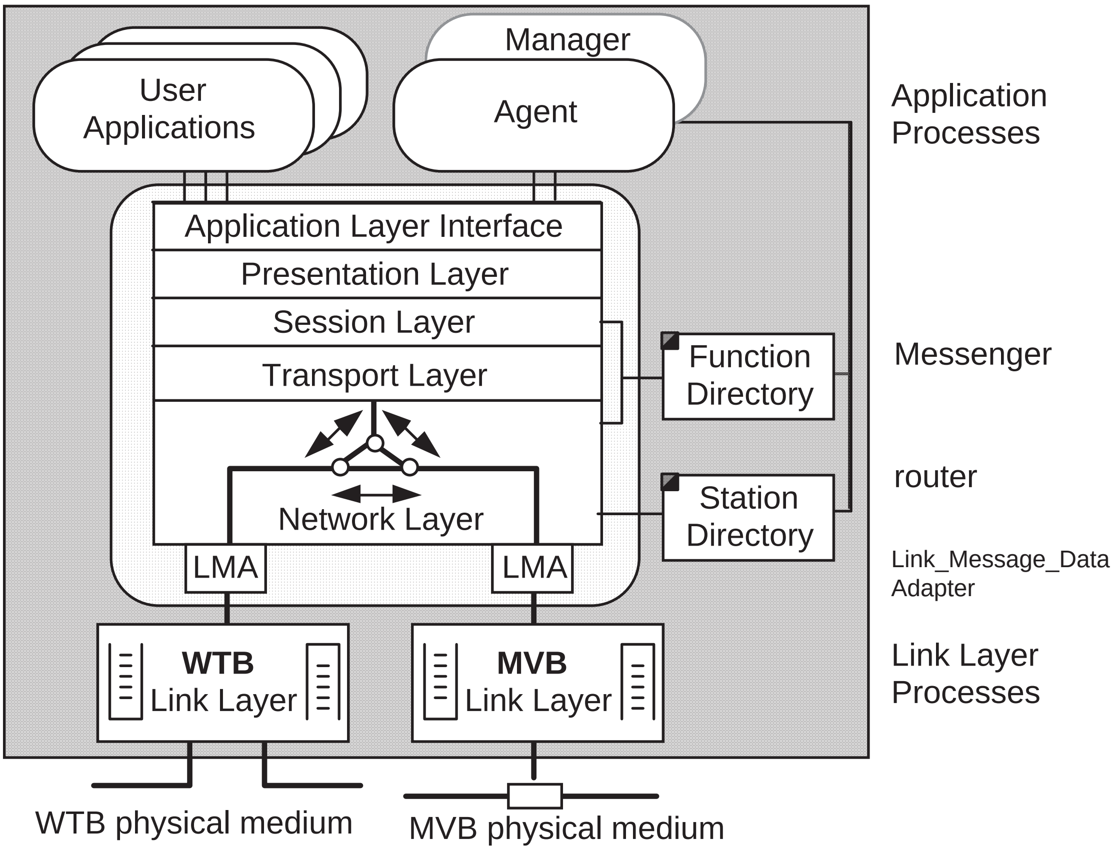
Figure 114 – Station terminale

NOTE La Couche réseau des stations terminales n'accède qu'à sa propre couche transport et à la couche liaison.

6.3.2.2 Station d'acheminement

Une station reliée à plusieurs bus, ou station d'acheminement, doit comporter une couche de liaison pour chaque bus. Les bus partagent les mêmes Protocoles en Temps Réel.

EXEMPLE La Figure 115 illustre une station d'acheminement reliée à la Couche de Liaison d'un MVB et d'un WTB.



Légende

Anglais	Français
User applications	applications utilisateur
manager	gestionnaire
agent	agent
application processes	processus d'application
application layer interface	interface de la couche application
presentation layer	couche présentation
session layer	couche session
transport layer	couche transport
network layer	couche réseau
messenger	messager
router	routeur
function directory	répertoire de fonctions
station directory	répertoire de stations
link layer processes	processus de la couche de liaison
link layer	couche de liaison
WTB physical medium	support physique WTB
MVB physical medium	support physique MVB

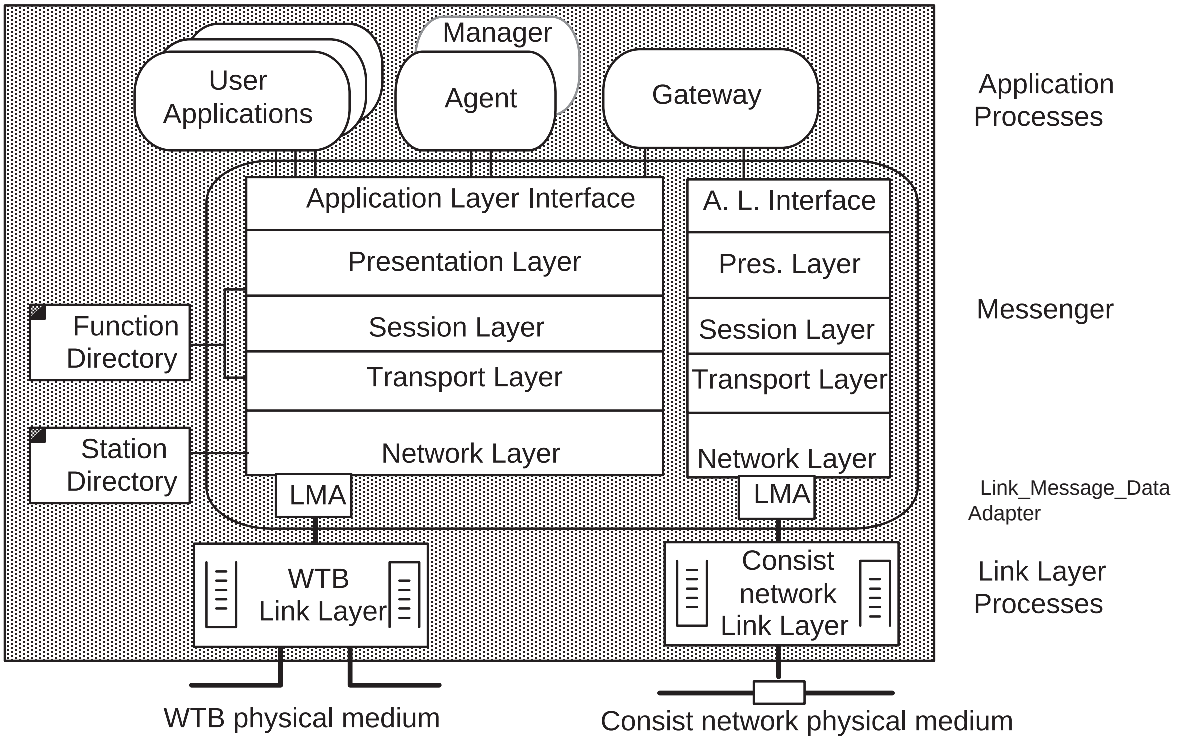
Figure 115 – Station d'acheminement entre le WTB et le MVB

NOTE La Couche Réseau d'une station terminale peut acheminer les paquets de bus en bus.

6.3.2.3 Station passerelle

Une station reliée à plusieurs bus, ou station passerelle, doit comporter une couche de liaison pour chaque bus. Le Réseau de Rame fait l'objet d'un protocole différent de WTB RTP. La station passerelle doit adapter les protocoles du Réseau de Rame au WTB RTP.

EXEMPLE La Figure 116 illustre une station passerelle reliée à la Couche de Liaison d'un Réseau de Rame et d'un WTB.



Légende

Anglais	Français
User applications	applications utilisateur
manager	gestionnaire
agent	agent
gateway	passerelle
application processes	processus d'application
application layer interface	interface de la couche application
presentation layer	couche présentation
session layer	couche session
transport layer	couche transport
network layer	couche réseau
messenger	messager
function directory	répertoire de fonctions
station directory	répertoire de stations
link layer processes	processus de la couche de liaison
WTB link layer	couche de liaison WTB
consist network link layer	couche de liaison de réseau de rame
WTB physical medium	support physique WTB
consist network physical medium	support physique du réseau de rame

Figure 116 – Station passerelle entre le WTB et le Réseau de Rame

6.3.2.4 Indicatif de station

Une station doit être identifiée par un Station_Id.

Le Station_Id identifie également l'Agent puisqu'il se peut qu'il n'y ait qu'un seul Agent et qu'un seul Messager par station.

NOTE L'Indicatif de station n'est pas nécessairement identique à une Device_Address. Une station d'acheminement comporte une Device_Address par bus auquel elle est reliée, mais ne comporte qu'un seul Station_Id. L'indicatif de station peut être modifié par la Gestion de Réseau, tandis que Device_Address est souvent câblée.

6.3.2.5 Indicatif de bus

Une station doit identifier chaque couche de liaison, c'est-à-dire chaque bus (réseau de rame ou bus de train) auquel elle est reliée par un Bus_Id.

Le nombre maximal de bus reliés à une station doit être 16.

NOTE Bus_Id ne précise pas le type de bus relié, mais il peut être utile de toujours préciser Bus_Id = 1 au bus de train.

6.3.2.6 Adresse de liaison

Une station doit identifier chaque dispositif auquel elle peut accéder sur l'un de ses bus par une Link_Address.

La Link_Address doit comprendre la concaténation du Bus_Id et de la Device_Address du dispositif.

NOTE La taille de la Device_Address dépend du bus.

6.3.3 Traitement des paquets de messages

Le traitement de paquets à l'intérieur d'une pile de protocole de communication est une question de mise en œuvre. Les procédures d'interface dans les Services de Messagerie spécifient un paquet de messages à l'aide d'un pointeur. C'est à la mise en œuvre d'utiliser ce pointeur pour envoyer les contenus de paquet par référence ou en les copiant. La structure même du paquet n'est pas spécifiée.

Ce paragraphe définit certaines procédures de traitement de paquets pour faciliter la portabilité et expliquer certaines procédures d'interface. Les paragraphes qui suivent n'impliquent pas une mise en œuvre particulière. Toute interface qui fournit la même sémantique est autorisée.

Cette interface n'a pas besoin d'être exposée et ne fait pas l'objet d'un Essai de Conformité.

6.3.3.1 Groupements de paquets

Tous les paquets utilisés par la couche transport, la couche réseau et la couche de liaison ont le même format afin d'assurer un accès cohérent à l'intérieur d'une station.

Pour envoyer les paquets par référence plutôt qu'en les copiant, une gestion de mémoire dynamique est nécessaire, sous forme de groupements de paquets. Un groupement de paquets est tout d'abord créé avec un certain nombre de paquets vides. Un utilisateur peut demander des paquets du groupement et les renvoyer au groupement après les avoir utilisés. A la longue, il convient que le débit net de paquets depuis et vers un groupement soit zéro.

Il existe plusieurs groupements de paquets, en général un couple pour chaque couche de liaison. Le groupement auquel un paquet appartient est son Propriétaire.

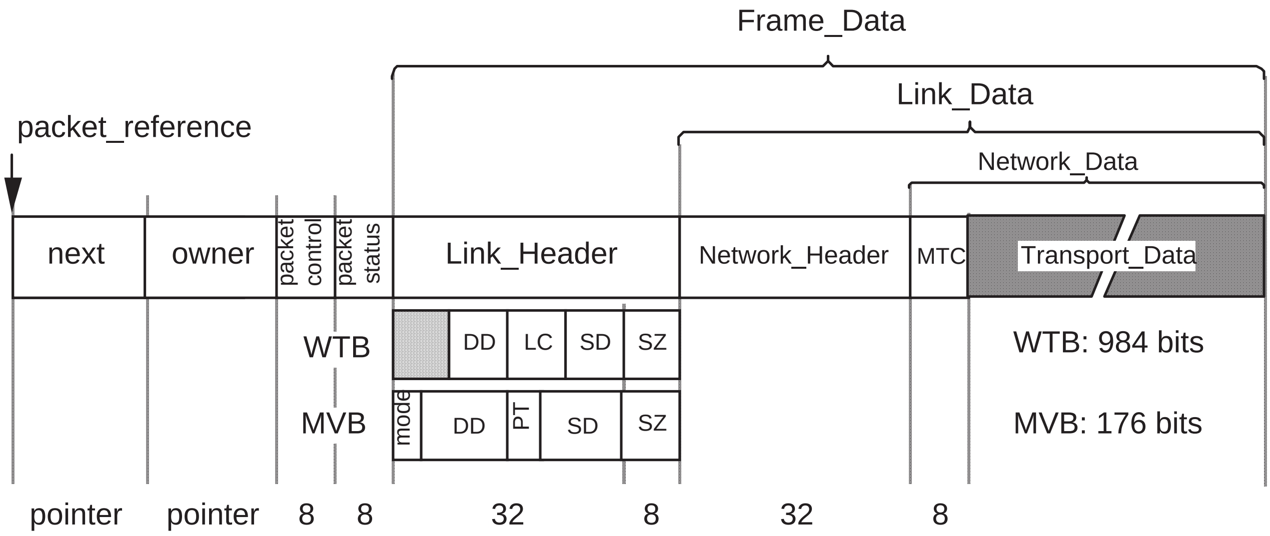
6.3.3.2 Type MD_PAQUET

Les paquets sont identifiés par un descripteur de paquet, qui en précise les données à envoyer ainsi que d'autres champs utilisés pour la gestion de paquets.

Un paquet est référencé par un pointeur de paquet, qui est unique au paquet concerné et qui pointe vers le descripteur de paquet qui est du type MD_PAQUET.

Définition	Ce type définit le format du paquet.
Syntaxe	`typedef void* MD_PACKET;`
Usage	Le format du MD_PAQUET n'est pas prescrit.

EXEMPLE La Figure 117 donne un exemple du MD_PAQUET.



Légende

Anglais	Français
next	suivant
owner	propriétaire
packet control	contrôle de paquet
packet status	état du paquet
pointer	pointeur

Figure 117 – Format de paquet

L'exemple de paquet illustré à la Figure 117 commence par les champs suivants:

– 'suivant': le premier champ du paquet est réservé à un pointeur vers un autre paquet. Cela permet la liaison des paquets en listes ou files d'attente.

– 'propriétaire': le deuxième champ identifie le propriétaire (groupement) du paquet. Ce champ est utilisé pour détruire un paquet après utilisation.

– 'contrôle de paquet' contient des informations de gestion que la couche de lLiaison peut lire mais ne peut modifier.

– 'état du paquet' est modifié par la couche de liaison et peut être lu par les autres couches.

Le reste du paquet contient la trame envoyée ou reçue sur le bus:

– le format de Link_Header est différent selon le bus (MVB, WTB ou autre). Il contient les Device_Address source et destinataire, parmi d'autres informations de contrôle spécifiques au bus.

– Le Link_Header ne concerne que la couche de liaison, il n'est pas analysé par la couche réseau, qui reçoit ces informations sous forme de paramètres.
– Le champ ‘taille’ s'applique aux Link_Data, à l'exception de la taille du champ lui-même.

L'état d'un paquet peut être:

- MD_PENDING ce paquet est marqué pour être envoyé;
- MD_FLUSHED ce paquet a été enlevé d'une file d'attente;
- MD_SENT ce paquet a été envoyé et peut être recyclé.

6.3.3.3 Type de procédure MD_GET_PACKET

Définition	extrait un nouveau paquet d'un groupement, attribue la valeur ‘groupement’ au champ propriétaire du paquet.	

Syntaxe	`typedef void ( * MD_GET_PACKET ) (`
	`void * * pool,`
	`MD_PACKET * * packet`
	`);`	

Entrée	groupement	identifie le groupement duquel le paquet est tiré
Sortie	paquet	Pointeur vers une structure de données dans laquelle le paquet est stocké.
Usage	Ce type de procédure est compatible avec la procédure LM_GET_PACK, qui peut être appelée directement.	

6.3.3.4 Type de procédure MD_PUT_PACKET

Définition	renvoie un seul paquet qui n'est plus utilisé au groupement spécifié par le champ propriétaire du paquet.	

Syntaxe	`typedef void ( * MD_PUT_PACKET ) (`
	`MD_PACKET * packet`
	`);`	

Entrée	paquet	Pointeur vers une structure de données dans laquelle le paquet peut être trouvé. Le groupement propriétaire est marqué dans le paquet.
Usage	Ce type de procédure est compatible avec la procédure LM_SEND_CONFIRM, qui peut être appelée directement.	

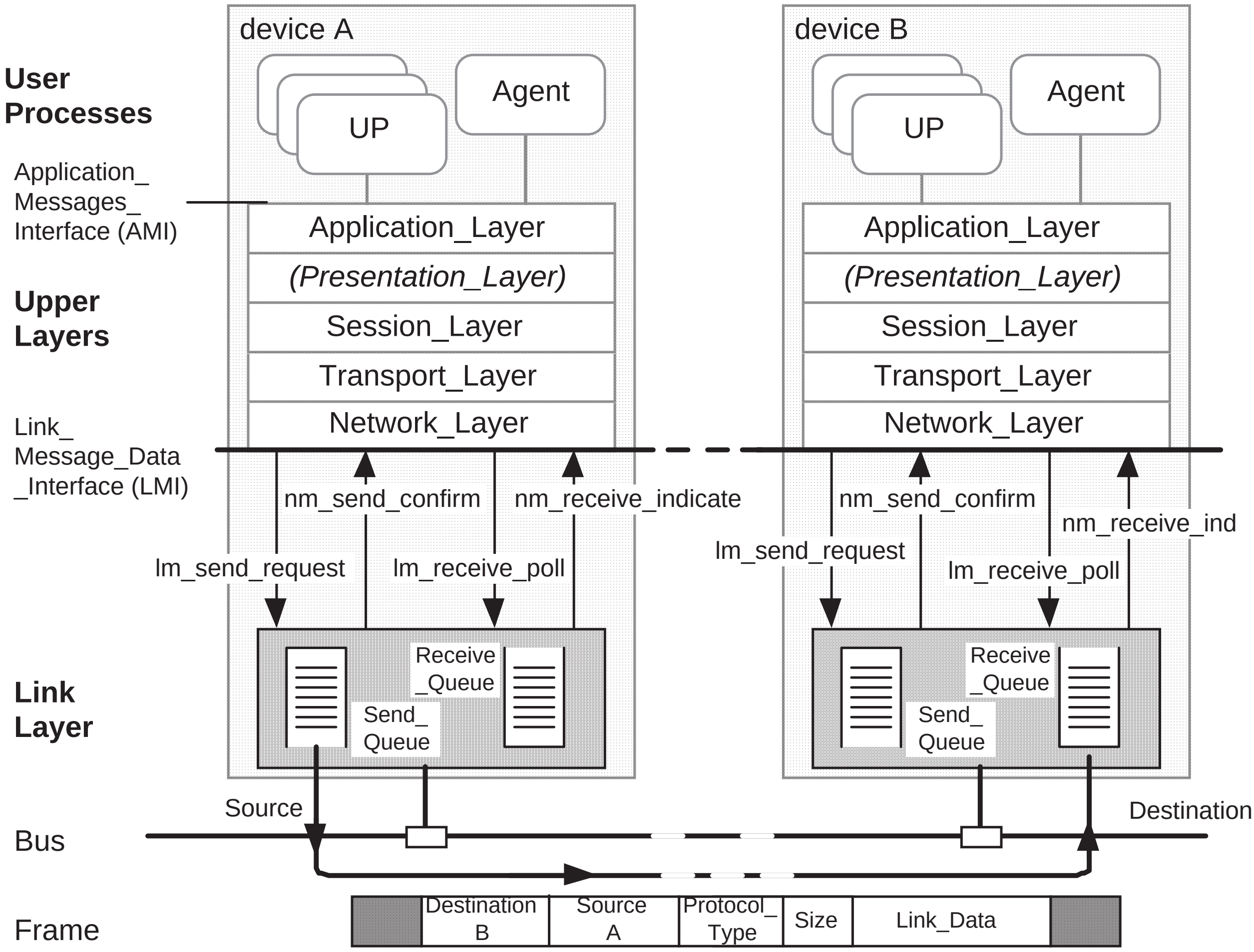
6.3.4 Couche de Liaison de Messagerie

6.3.4.1 Objet

Les Services de Messagerie sont fournis sur différents bus: WTB, MVB ou autres, y compris les bus parallèles, les liaisons série, les boîtes à lettres mémoire ou les bus capteurs.

A cet effet, la couche de liaison de l'un de ces bus est censée fournir un ensemble de services de base spécifiés dans ce paragraphe.

En général, la couche de liaison d'un dispositif coopère avec la couche de liaison d'un autre dispositif pour échanger des paquets entre un dispositif émetteur et un dispositif destinataire sur le même bus (voir la Figure 118).



Légende

Anglais	Français
User processes	processus utilisateur
device	dispositif
agent	agent
upper layers	couches supérieures
link layer	couche de liaison
source	émetteur
destination	destinataire
bus	bus
frame	trame
size	taille

Figure 118 – Transmission de données de couche de liaison

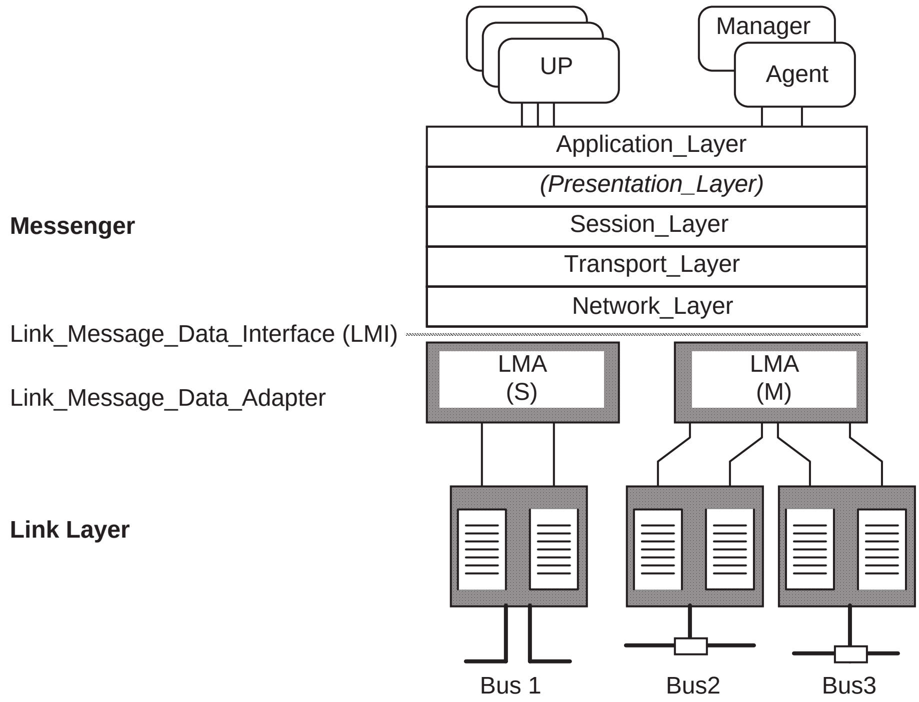
Le processus de couche de liaison réalise la transmission sur le bus. Il peut s'exécuter indépendamment des Applications et du Messager.

La mise en œuvre de la couche de liaison n'est pas spécifiée. Par conséquent, les services de la couche de liaison sont spécifiés dans une forme générale qui doit se trouver sur chaque bus relié au Messager.

6.3.4.2 Structure de la Couche de Liaison

Une couche de liaison doit fournir deux files d'attente, une Send_Queue dans laquelle la couche réseau place les paquets à transmettre et une Receive_Queue dans laquelle la couche réseau récupère les paquets reçus du bus.

Une station (d'acheminement) peut comporter plusieurs couches de liaison, une pour chaque bus relié (voir la Figure 119).



Légende

Anglais	Français
messenger	messager
manager	gestionnaire
agent	agent
link layer	couche de liaison

Figure 119 – Interface LMI

Les différences de mise en œuvre des couches de liaison sont masquées dans le LMA (Link_Message_Data_Adapter).

NOTE Un LMA peut prendre en charge un seul bus (Type S) ou plusieurs bus (Type M).

6.3.4.3 Caractéristiques de la couche de liaison

6.3.4.3.1 Adresse de dispositif

Chaque dispositif d'un bus doit être identifié de manière unique par sa Device_Address.

Device_Address 0 doit identifier la couche de liaison locale et ne doit pas être allouée à un dispositif particulier.

La Device_Address la plus élevée ('11111111'B pour une adresse de dispositif à 8 bits, par exemple) doit indiquer une diffusion vers tous les dispositifs présents sur le bus et ne doit pas être allouée à un dispositif particulier.

La couche de liaison doit inclure sa propre Device_Address comme Source_Device dans tous les paquets qu'elle transmet, mais ne doit pas utiliser l'adresse de diffusion spécifique au bus comme Source_Device.

La couche de liaison doit inclure la Device_Address du dispositif distant ou l'adresse de diffusion spécifique au bus comme Destination_Device dans tous les paquets qu'elle transmet.

NOTE 1 Le format de la Device_Address dépend du bus (WTB, MVB ou autre).

NOTE 2 Un dispositif d'acheminement possède une Device_Address pour chaque bus relié.

6.3.4.3.2 Type de protocole

La couche de liaison pour les messages doit se présenter sous la forme d'une paire de files d'attente (mémoires tampons).

Si le même dispositif prend en charge d'autres protocoles, un Protocol_Type (PT) dans chaque trame doit sélectionner les Send_Queues et les Receive_Queues utilisées pour les Protocoles en Temps Réel.

NOTE Le Protocol_Type joue un rôle semblable à celui d'un Point d'Accès de Service de Liaison dans l'ISO/CEI 7498.

6.3.4.3.3 Priorités

Les priorités ne sont pas distinguées au niveau de la couche de liaison.

6.3.4.3.4 Vidage

La couche de liaison d'un nœud doit comprendre des dispositions de vidage de tous les paquets.

NOTE Le vidage est nécessaire si un nœud reçoit une nouvelle Topographie.

6.3.4.3.5 Durée de vie d'un paquet

La Couche de Liaison doit comprendre des dispositions visant à limiter la durée de vie des paquets dans sa Receive_Queue ou sa Send_Queue à une valeur inférieure à PACK_LIFE_TIME.

PACK_LIFE_TIME doit être de 5,0 s.

NOTE La durée de vie peut être limitée par une temporisation de file d'attente (vidage de la file d'attente) ou par l'invalidation individuelle du paquet dans la file d'attente.

6.3.4.3.6 Protocole de couche de liaison

Le protocole de couche de liaison doit être sans connexion, c'est-à-dire ne fonctionner qu'avec des datagrammes.

La couche de liaison ne doit pas répéter automatiquement les trames perdues.

La couche de liaison doit enregistrer les erreurs de transmission pour la gestion de réseau.

6.3.4.4 Couche de liaison du réseau de rame, exemple de MVB

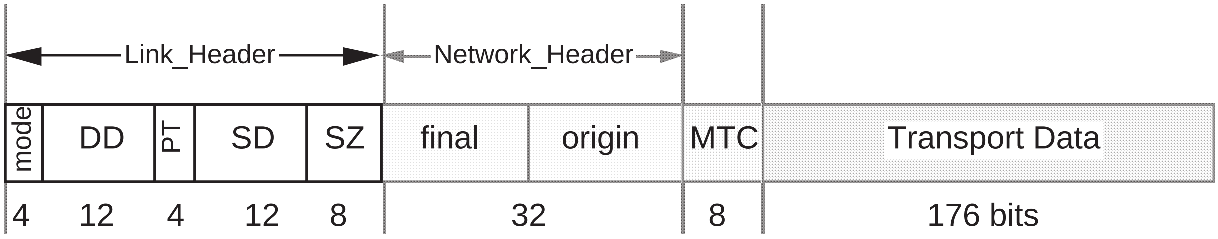
Le réseau de rame peut être mis en œuvre par différents bus, comme le MVB, par exemple (voir la CEI 61375-3-1) qui sert de référence.

La Device_Address d'un réseau de rame ne doit pas changer pendant le fonctionnement normal.

Si le MVB fait office de réseau de rame, les spécifications supplémentaires suivantes s'appliquent:

- la Device_Address à 12 bits doit identifier les dispositifs émetteur et destinataire;
- le champ Mode doit spécifier un envoi point à point (‘0001’B) ou une transmission en diffusion (‘1111’B). Dans le deuxième cas, la Device_Address est ‘indifférente’;
- si la Device_Address la plus élevée (‘111111111111’B) est précisée, le mode de transmission en diffusion doit être sélectionné;
- le Protocol_Type ‘1000’B à 4 bits doit identifier les Protocoles en Temps Réel du Réseau Embarqué de Train.

EXEMPLE La Figure 120 illustre le format d'une trame Message_Data sur le MVB.



Légende

Anglais	Français
final	final
origin	origine
transport data	données de transport

Figure 120 – Exemple de trame Message_Data sur le MVB

6.3.4.5 Couche de liaison du bus de train

Le bus de train peut être mis en œuvre par différents bus et plus particulièrement par le WTB qui sert de référence. La distinction est faite entre les bus de train à composition variable (le WTB, par exemple) et les bus de train à composition fixe (le MVB, par exemple).

6.3.4.5.1 Bus de train à composition variable

Dans un bus de train à composition variable, dont les adresses de nœud peuvent changer de manière dynamique, la couche de liaison doit avertir sa couche réseau d'un changement et doit lui fournir le Topo_Counter, un compteur à 6 bits qui est incrémenté (modulo 64) à chaque changement de composition.

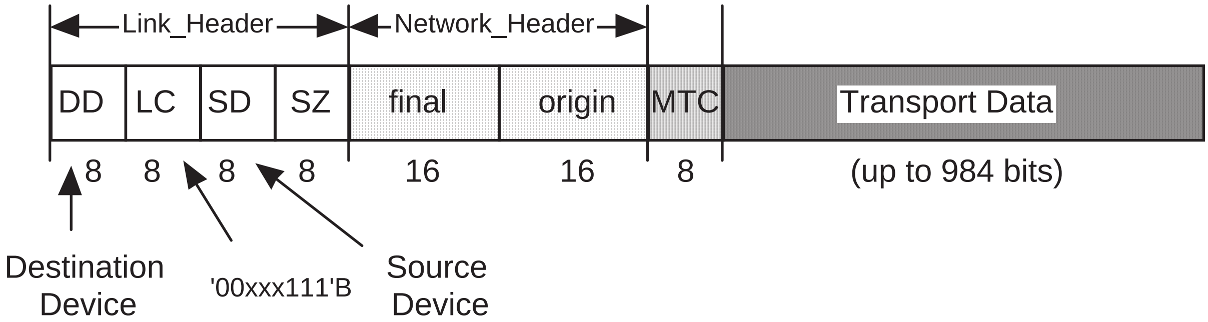
NOTE 1 Le bus de train peut fournir des informations supplémentaires (la Topographie, par exemple) par l’intermédiaire des Services de Gestion de couche de liaison, spécifiées dans le WTB. Bien que les services de messagerie ne tiennent compte que du Topo_Counter, l'application a besoin de la Topographie pour sélectionner les adresses de nœud correctes en fonction du type de rame. La mise en correspondance de la rame et des adresses de nœud est une question d'application.

NOTE 2 Dans la mesure où une application peut ne pas être informée de la distribution d'une nouvelle Topographie depuis la dernière lecture de l'ancienne, le Topo_Counter est utilisé pour valider les informations relatives à la Topographie.

Si le WTB fait office de bus de train, les spécifications supplémentaires suivantes s'appliquent.

– Les nœuds sont identifiés par les adresses de nœud du WTB que l’inauguration attribue par l’intermédiaire du Node_Address à 8 bits.
– Le WTB utilise la Destination_Device = ‘11111111’B comme adresse de diffusion.
– Le Protocol_Type de la couche de liaison pour les Protocoles en Temps Réel est identifié par Link_Control = ‘x0xxx111’B (Message_Data).

EXEMPLE La Figure 121 illustre le format de trame du WTB.



Légende

Anglais	Français
final	final
origin	origine
transport data	données de transport
destination device	dispositif destinataire
source device	dispositif émetteur
up to	jusqu'à

Figure 121 – Exemple de trame Message_Data sur le WTB

6.3.4.6 Interface de couche de liaison de Données de Messagerie (LMI)

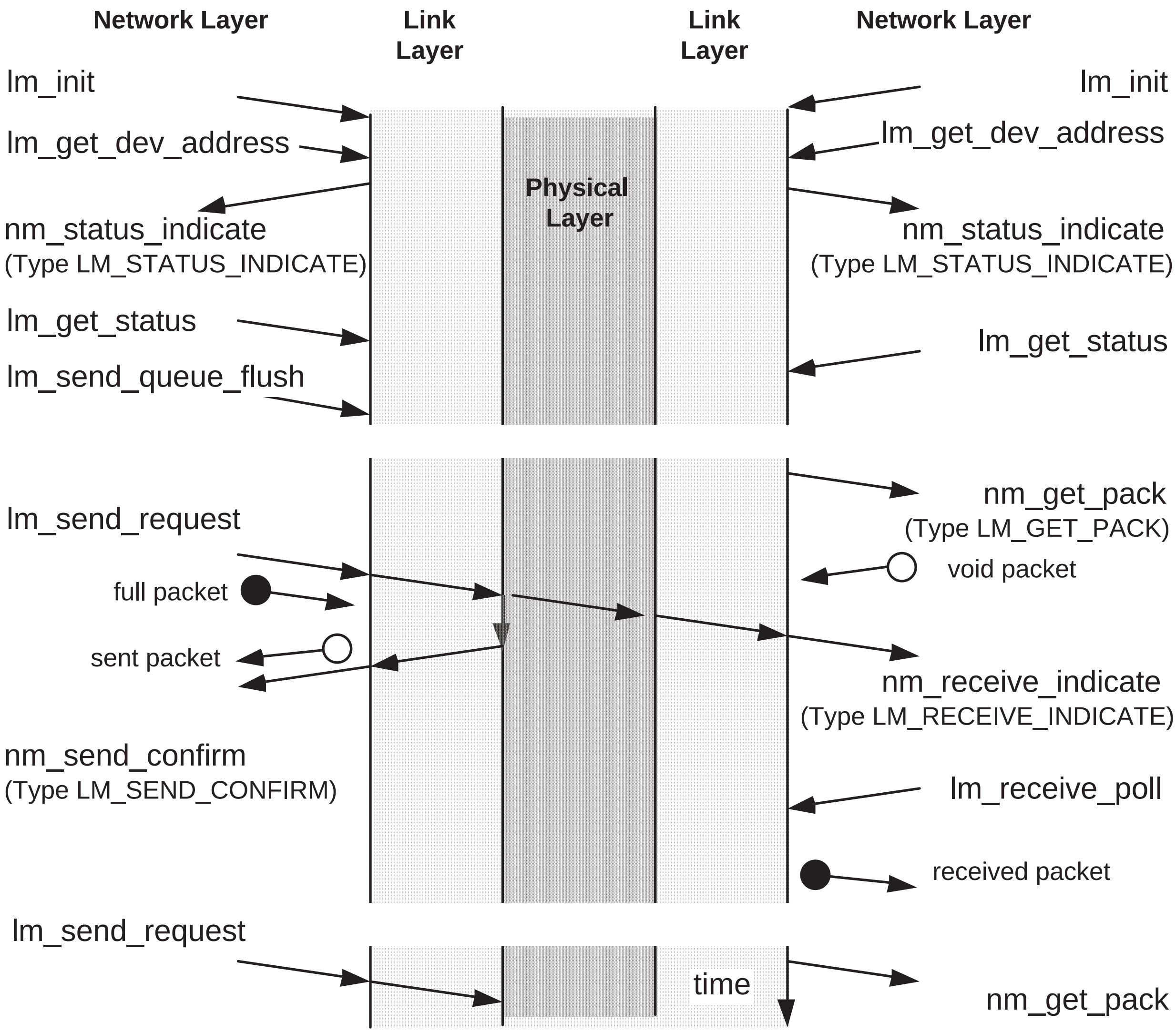
La LMI définit les services qu'une couche de liaison propose à la couche réseau.

Cette interface ne fait pas l'objet d'Essais de Conformité. Elle est spécifiée dans les paragraphes suivants pour faciliter le portage.

Les paragraphes qui suivent n'impliquent pas une mise en œuvre particulière. Toute interface qui fournit la même sémantique est autorisée.

6.3.4.6.1 Primitives LMI

La LMI doit fournir les primitives illustrées à la Figure 122, répertoriées dans le Tableau 20 et décrites dans les paragraphes suivants.



Légende

Anglais	Français
network layer	couche réseau
link layer	couche de liaison
physical layer	couche physique
full packet	paquet complet
sent packet	paquet émis
void packet	paquet vide
time	durée

Figure 122 – Primitives LMI

NOTE Les procédures LMI, les types de la couche de liaison et les primitives de la couche réseau sont respectivement précédées de lm_, de LM_ et nm_.

La couche de liaison appelle les procédures d'indication de la couche réseau (précédées de nm_) qui étaient souscrites et dont le type est défini par la couche de liaison (précédé de LM_).

Tableau 20 – Primitives LMI

Nom	Signification
MD_RESULT	Résultat d'une procédure
lm_send_request	Demande d'envoi d'un paquet
LM_SEND_CONFIRM	Indication qu'un paquet a été envoyé
LM_GET_PACK	Reçoit un paquet vide
LM_RECEIVE_INDICATE	Indique qu'un paquet a été reçu
lm_receive_poll,	Interrogation de réception de paquets
LM_STATUS_INDICATE,	Indique un changement d'état
lm_get_status,	Extrait l'état de la couche de liaison
lm_send_queue_flush,	Vide la Send_Queue
lm_init	Initialise la couche de liaison
lm_get_dev_address.	Lit la Device_Address

6.3.4.6.2 Type MD_RESULT

Définition	Une procédure LMI qui renvoie une valeur doit la coder de la manière suivante:

Syntaxe	`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;`

6.3.4.6.3 Procédure lm_send_request

Définition	Ajoute le Link_Header à un paquet et l'insère dans la Send_Queue. Si la couche de liaison accepte un paquet pour transmission, elle définit son état à MD_PENDING.	

Syntaxe	`MD_RESULT            lm_send_request`
	`                     (`
	`    ENUM8            bus_id`
	`    UNSIGNED32       source,`
	`    UNSIGNED32       destination,`
	`    MD_PACKET *      packet;`
	`                     )`	

Entrée	bus_id	Sélectionne l'une des 16 différentes couches de liaison.
	source	Device_Address du dispositif émetteur. Si 'source' est nulle, la couche de liaison inclut sa Device_Address dans le paquet.
	destination	Device_Address du dispositif destinataire ou une adresse de diffusion si Device_Address la plus haute possible a été définie.
	paquet	Pointeur vers une structure de données qui contient le paquet. Le paquet lui-même contient sa taille et son état.
Renvoi		toute valeur de MD_RESULT

6.3.4.6.4 Type de procédure LM_SEND_CONFIRM

Définition	La couche de liaison appelle une procédure de ce type pour renvoyer le paquet au groupement une fois qu'il a été transmis ou qu'il n'est plus utile.	
Syntaxe	`typedef void ( * LM_SEND_CONFIRM ) ( MD_PACKET * packet );`	
Entrée	packet	Pointeur vers une structure de données qui contient le paquet transmis ou vidé. Le paquet lui-même contient sa taille et son état.
Usage	1 – La procédure ‘nm_get_pack’ (du type LM_GET_PACK) a déjà été souscrite dans la procédure ‘lm_init’. 2 – La couche de liaison met l'état du paquet à MD_SENT (transmission réussie) ou à MD_FLUSHED (file d'attente vidée) avant d'appeler ‘nm_send_confirm’.	

```
typedef void       ( * LM_SEND_CONFIRM )
                   (
    MD_PACKET *    packet
                   );
```

6.3.4.6.5 Type de procédure LM_GET_PACK

Définition	Une procédure de ce type est appelée pour demander un paquet libre d'un groupement. Cette procédure, si elle aboutit, fournit un seul paquet dont le champ propriétaire a la valeur du paramètre propriétaire.	
Syntaxe	`typedef void ( * LM_GET_PACK ) ( void * * owner, MD_PACKET * * packet );`	
Entrée	owner	identifie le groupement d’origine du paquet.
	packet	Pointeur vers une structure de données dans laquelle le paquet est stocké.
Usage	1 – La procédure ‘nm_get_pack’ (du type LM_GET_PACK) a déjà été souscrite dans la procédure ‘lm_init’. 2 – La couche de liaison doit préciser comme propriétaire uniquement le groupement qui lui a été alloué lors de l'initialisation.	

```
typedef void       ( * LM_GET_PACK )
                   (
    void * *       owner,
    MD_PACKET * *  packet
                   );
```

6.3.4.6.6 Type de procédure LM_RECEIVE_INDICATE

Définition	A la réception d'un paquet, la couche de liaison doit appeler une procédure de ce type.	

Syntaxe	`typedef void ( * LM_RECEIVE_INDICATE )`
	`(`
	`ENUM8 bus_id`
	`);`	

Entrée	bus_id	Bus_Id (0..15)
Usage	1 – La procédure 'nm_receive_indicate' (du type LM_RECEIVE_INDICATE) a déjà été souscrite dans la procédure lm_init. 2 – Cette procédure peut être NULL si l'interrogation (lm_receive_poll) est utilisée. 3 – Cette procédure d'indication est appelée à partir d'un programme de service d'interruption, mais elle est autorisée à adresser des appels au noyau. Elle est destinée à réveiller un Messager.	

6.3.4.6.7 Procédure lm_receive_poll

Définition	Lit un paquet reçu de la Receive_Queue et envoie la référence du paquet à la couche réseau.	

Syntaxe	`MD_RESULT lm_receive_poll`
	`(`
	`ENUM8 bus_id,`
	`unsigned * source,`
	`unsigned * destination,`
	`MD_PACKET * * packet`
	`);`	

Entrée	bus_id	Bus_Id (0..15) de cette couche de liaison
Renvoi		toute valeur de MD_RESULT
Sortie	source	Device_Address du dispositif émetteur
	destination	Device_Address du dispositif destinataire ou adresse de 'diffusion'.
	packet	Pointeur vers une structure de données qui contient le paquet reçu ou bien NULL si la file d'attente est vide. Le paquet lui-même contient sa taille.
Usage	1 – Le descripteur de paquet peut être réutilisé une fois que cette procédure a été appelée. 2 –Si le résultat de cette procédure est MD_OK, la Receive_Queue est vidée d'une position et les autres arguments de cette procédure deviennent significatifs. 3 – Si cette procédure est appelée quand la Receive_Queue est vide, elle renvoie MD_REJECT comme résultat et NULL comme 'paquet'.	

6.3.4.6.8 Type de procédure LM_STATUS_INDICATE

Définition	Si une exception se produit, la couche de liaison doit appeler une procédure de ce type pour signaler ce fait.	

Syntaxe	`typedef void ( * LM_STATUS_INDICATE )`
	`(`
	`ENUM8 bus_id,`
	`MD_RESULT status`
	`);`	

Entrée	bus_id	Bus_Id (0..15)
	status	MD_OK (fonctionnement normal) MD_REJECT (bus non disponible) MD_INAUGURATION (inauguration du bus de train)
Usage	1 – La procédure 'nm_status_indicate' (de type LM_STATUS_INDICATE) a déjà été souscrite par la procédure 'lm_init'. 2 – L'état, de type MD_RESULT, est un paramètre d'entrée.	

6.3.4.6.9 Procédure lm_get_status

Définition	Récupère des informations détaillées sur la couche de liaison.	

Syntaxe	`MD_RESULT lm_get_status`
	`(`
	`ENUM8 bus_id,`
	`BITSET8 selector,`
	`BITSET8 reset,`
	`BITSET8 * status`
	`);`	

Entrée	bus_id	Bus_Id (0..15)
	selector	sélectionne les bits concernés renvoyés dans l'état Chaque information est représentée par un seul bit dont la valeur est fixée par la couche de liaison et réinitialisée par l'utilisateur du service. Les trois bits suivants sont définis: MD_RECEIVE_ACTIVE = 1 activé lorsqu'une trame correcte est reçue de la part du bus. MD_SEND_ACTIVE = 2 activé lorsqu'une trame a été transmise. MD_RECEIVE_OVERFLOW = 4 activé lorsqu'une trame reçue est perdue parce qu'il n'y a plus de paquet libre.
	reset	sélectionne les bits à effacer par la suite
Renvoi		toute valeur de MD_RESULT
Sortie	status	renvoie la valeur des bits d'état sélectionnés.
Usage	La Couche Réseau est censée réinitialiser le bit MD_SEND_ACTIVE lorsqu'il lit l'état de la couche de liaison d'un Réseau de Rame.	

6.3.4.6.10 Procédure lm_send_queue_flush

Définition	Vide la Send_Queue dans la couche de liaison et définit l'état du paquet à MD_FLUSHED. Appelle nm_send_confirm une fois que chaque paquet a été inséré par lm_send_request et n'a pas encore été envoyé.	
Syntaxe	`MD_RESULT lm_send_queue_flush ( ENUM8 bus_id /* optional */ );`	
Entrée	bus_id	Bus_Id (0..15)
Renvoi		toute valeur de MD_RESULT

```
MD_RESULT         lm_send_queue_flush
                  (
    ENUM8         bus_id  /* optional */
                  );
```

6.3.4.6.11 Procédure lm_init

Définition	Initialise la couche de liaison, vide la Send_Queue et souscrit les procédures d'indication de la couche réseau.	
Syntaxe	(voir ci-dessous)	
Entrée	bus_id	Bus_Id (0..15)
	nm_receive_indicate	procédure de la couche réseau appelée par la couche de liaison chaque fois qu'elle reçoit un paquet.
	get_packet	procédure du groupement de paquets appelée par la couche de liaison chaque fois qu'elle a besoin d'un paquet vide.
	owner	groupement de paquets utilisé par la couche de liaison pour en extraire les paquets.
	put_pack	procédure du groupement de paquets appelée par la couche de liaison pour détruire un paquet.
	nm_status_indicate	procédure de la Couche Réseau appelée par la couche de liaison pour signaler une exception (une inauguration, par exemple).
Renvoi		toute valeur de MD_RESULT

```
MD_RESULT              lm_init
                       (
ENUM8                  bus_id,
LM_RECEIVE_INDICATE    nm_receive_indicate,
LM_GET_PACK            nm_get_pack,
void * *               owner,
LM_SEND_CONFIRM        nm_send_confirm,
LM_STATUS_INDICATE     nm_status_indicate
                       );
```

6.3.4.6.12 Procédure lm_get_dev_address

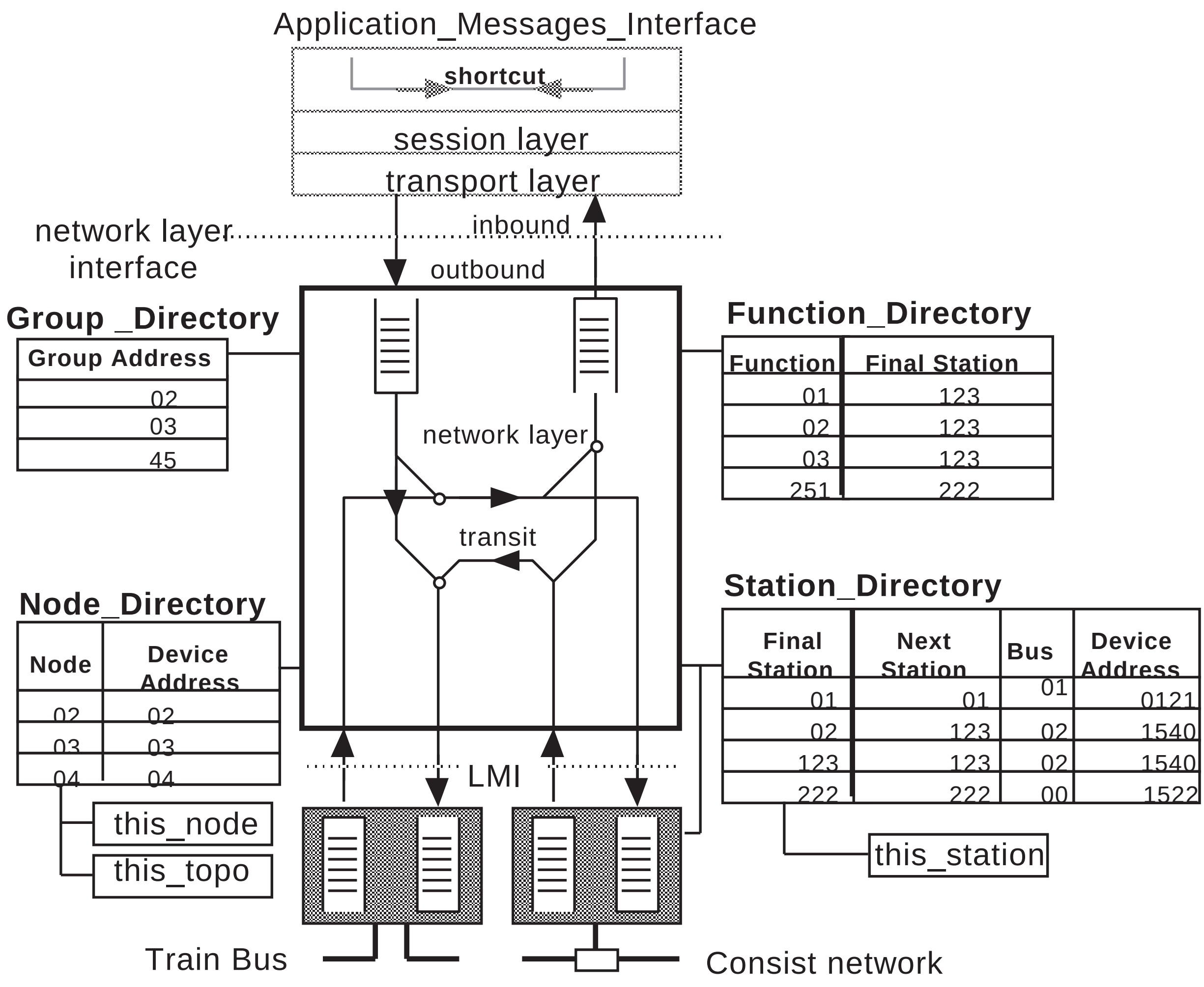
Définition	Lit la Device_Address correspondant à la couche de liaison spécifiée.	
Syntaxe	`MD_RESULT lm_get_dev_address ( ENUM8 bus_id, UNSIGNED * device_address );`	
Entrée	bus_id	Bus_Id (0..15)
Renvoi		toute valeur de MD_RESULT
Sortie	device_address	Device_Address de ce dispositif sur cette couche de liaison
Usage	Sur une couche de liaison du bus de train, la Device_Address est extraite et la Node_Address peut en être déduite.	

6.3.5 Couche de Réseau de Messagerie

6.3.5.1 Objet

La couche réseau achemine les paquets (voir la Figure 123):

- depuis la couche transport de sa station vers une couche de liaison (paquets sortants); ou
- depuis l'une de ses couches de liaison vers la couche transport (paquets entrants);
- entre deux couches de liaison dans un Nœud d'acheminement (paquets de transit).



Légende

Anglais	Français
shortcut	raccourci
session layer	couche session
transport layer	couche transport
inbound	entrant
outbound	sortant
network layer interface	interface de la couche réseau
network layer	couche réseau
final station	station finale
next station	station suivante
bus	bus
device address	adresse de dispositif
train bus	bus de train
consist network	réseau de rame
node	nœud
function	fonction
group address	adresse de groupe

Figure 123 – Couche réseau sur un nœud

La couche réseau achemine les paquets depuis une station d'origine vers une station finale.

Pour cela, la couche réseau utilise la mise en correspondance fournie par plusieurs répertoires:

a) le répertoire de stations,
b) le répertoire de fonctions,
c) le répertoire de groupes, et
d) le répertoire de nœud.

La couche réseau est sans connexion.

6.3.5.2 Répertoires

6.3.5.2.1 Répertoire de stations (en option)

La couche réseau doit établir une correspondance entre un indicatif de station et l'adresse de couche de liaison de cette station, et inversement.

Dans une station terminale, cette mise en correspondance peut être obtenue en allongeant le Station_Id par un préfixe pour obtenir la Device_Address (correspondance simple).

Dans une station d'acheminement, cette mise en correspondance doit être réalisée par un répertoire de stations structuré de la manière suivante:

Station_Id	indicatif de station (clé vers le répertoire de stations)
Next_Station_Id	indicatif de la station suivante par laquelle la station peut être atteinte.
Bus_Id	couche de liaison par laquelle la Next_Station peut être atteinte.
Device_Address	Device_Address sur le bus par laquelle Next_Station peut être atteinte.

NOTE 1 Le répertoire de stations permet de construire le Link_Header d'un paquet sortant et d'identifier l'adresse source de la couche de liaison d'un paquet entrant. En effet, il convient que la Couche Transport ne traite pas les adresses spécifiques au bus de taille arbitraire, mais seulement les indicatifs de station.

NOTE 2 La Station_Id et la Next_Station sont identiques lorsque la station peut être atteinte directement. Elles sont différentes lorsque les paquets sont acheminés entre deux réseaux de rame ou entre un réseau de rame et un bus capteur.

NOTE 3 Les primitives d'accès du répertoire de stations sont définies dans l'ALI.

6.3.5.2.2 Répertoire de fonctions

La Couche Réseau doit mettre en œuvre le répertoire de fonctions, qui établit une correspondance entre un Function_Id et une Station_Id, dont la structure est la suivante:

Function_Id	indicatif de fonction (clé vers le répertoire de fonctions)
Station_Id	indicatif de station (clé vers le répertoire de stations)

NOTE 1 Le Function_Directory appartient à la couche réseau, mais il est accessible à toutes les couches, à l'exception de la couche de liaison.

NOTE 2 Les primitives d'accès du répertoire de fonctions sont définies dans l'ALI.

6.3.5.2.3 Répertoire de groupes

La Couche Réseau d'une station qui participe à une distribution doit indiquer le Groupe auquel la station appartient par un répertoire de groupes dont la structure est la suivante:

Membership	liste de groupes

NOTE Les primitives d'accès du répertoire de groupes sont définies dans l'ALI.

6.3.5.2.4 Répertoire de nœud (en option)

La Couche Réseau d'un nœud doit établir une correspondance entre la Node_Address et la Device_Address sur le bus de train.

Lorsque le WTB est utilisé comme un bus de train, une correspondance simple doit être utilisée, obtenue en préfixant la Node_Address à 6 bits de deux ‘0’ pour former une Node_Address de WTB à 8 bits. Le répertoire de nœud est en lecture seule, car son contenu ést défini par l’inauguration.

Dans des compositions de train fixes où cette correspondance simple n'est pas adaptée, une correspondance doit être établie entre la Node_Address et la Device_Address à l’aide d’un répertoire de nœud dont la structure est la suivante:

Node_Address	Node Address (clé vers le répertoire de nœud)
Bus_Id	couche de liaison correspondant au bus de train
Device_Address	adresse de dispositif spécifique au bus de train

NOTE Les primitives d'accès du répertoire de nœud sont définies dans l'ALI.

6.3.5.3 Constantes et variables de la couche réseau

Valeurs distinctes de Station_Id ou Next_Station:

Station_Id =	Signification
AM_SAME_STATION	0 (constante)
AM_UNKNOWN	255 (constante)
this_station	indicatif à 8 bits de cette station. Si aucun indicatif n’a été attribué à une station, this_station = AM_UNKNOWN.
final_station	station indiquée dans la Network_Address finale
origin_station	station indiquée dans la Network_Address d’origine
next_station	station par laquelle la couche réseau transmet un paquet ou de laquelle elle l'a reçu.

Valeurs distinctes du Nœud (toutes ces valeurs sont des UNSIGNED6):

Nœud =	Signification
AM_SAME_NODE	0 (constante)
this_node	Node_Address à 6 bits de ce Nœud
AM_ANY_TOPO	0 (constante), toute valeur de Topo_Counter étant admise
origin_node	Node_Address à 6 bits dans la Network_Address d'origine
final_node	Node_Address à 6 bits de la Network_Address finale
this_topo	valeur du Topo_Counter de cette station, enregistrée par am_set_current_tc.
my_topo	valeur du Topo_Counter pour cette conversation, indiquée par l'application
packet_topo	valeur du Topo_Counter transportée dans le paquet.

Procédures diverses:

multicast	renvoie vrai si l'adressage par groupe (distribution) est utilisé
member (group)	renvoie vrai si le Nœud appartient au groupe et si la distribution est utilisée
fundi ()	renvoie le Station_Id indiqué par le Function_Directory
stadi ()	renvoie la Link_Address indiquée par le Station_Directory.

6.3.5.4 Network_Address

6.3.5.4.1 Format

La couche réseau reçoit de sa couche transport ou d'une des couches de liaison une Network_Address finale avec chaque paquet.

La couche réseau utilise cette Network_Address finale pour générer le Link_Header et le Network_Header d'un paquet qu'il transmet.

La couche réseau lit le Link_Header et le Network_Header d'un paquet entrant et les convertit en une Network_Address d'origine pour sa couche transport.

La Figure 124 illustre un codage convenable de la Network_Address finale et d'origine.

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

Network_Address finale

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

Network_Address d'origine

NOTE La figure ne représente pas un format de transmission, mais un format d'interface.

Figure 124 – Codage de la Network_Address

6.3.5.4.2 Système ou Utilisateur (fsu/osu)

Le bit de poids fort du premier octet d'une Network_Address, fsu ou osu, spécifie s'il est à:

- 1: une System_Address: l'octet identifie une station, ou s'il est à
- 0: une User_Address: l'octet suivant identifie une fonction.

Le bit 'osu' de l'Origin_Address doit avoir la même valeur que le bit 'fsu' de la Final_Address.

6.3.5.4.3 Groupe ou Individuel (fgi/ogi)

Le second bit de poids fort du premier octet de la Network_Address finale spécifie s'il est à:

- 1: une adresse de groupe ou, s'il est à
- 0: une adresse individuelle

Les bits fgi et ogi de l'Origin_Address doivent avoir la même valeur que la Final_Address.

6.3.5.4.4 Nœud ou Groupe

Les 6 bits origin_node représentent:

- une Node_Address unique du nœud d'origine.

Les 6 bits final_node_or_group représentent:

- une Group_Address, si le bit fgi spécifie un groupe,
- sinon, une Node_Address unique.

6.3.5.4.5 Fonction ou Station

Le deuxième octet d'une Network_Address contient:

- un indicatif de station si la valeur 'System' est attribuée à fsu/osu;
- un indicatif de fonction si la valeur 'User' est attribuée à fsu/osu.

6.3.5.4.6 Link_Address

Le troisième octet, ‘next_station’, identifie la station reliée au même Nœud auquel un paquet doit être envoyé ou duquel un paquet a été reçu (il ne s’agit pas nécessairement de la station d’origine ou de la station finale).

La couche réseau obtient la Link_Address correspondante (Bus_Id et Device_Address) de son Station_Directory. Sinon, pour un paquet entrant, elle peut déduire le Station_Id de la Next_Station à partir de la Device_Address et du Bus_Id.

‘next_station’ peut également indiquer la station elle-même (AM_SAME_STATION) ou une station inconnue (AM_UNKNOWN).

Aucune next_station n’est définie sur un Nœud qui transmet sur le bus de train (AM_UNKNOWN). Cependant, si le Nœud utilise un répertoire de nœud, la Next_Station est l'entrée vers ce répertoire.

6.3.5.4.7 Compteur de topographie

Le quatrième octet de la Network_Address finale et d'origine contient le Topo_Counter.

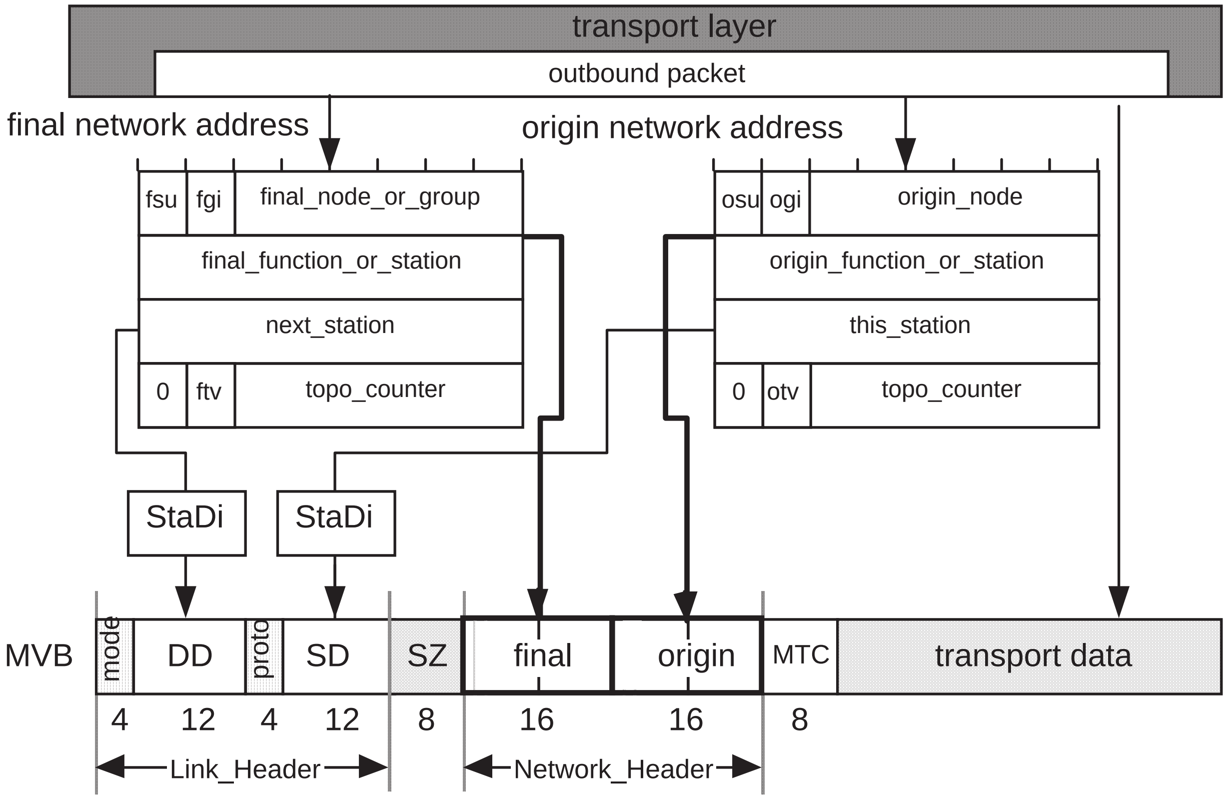
Le bit de poids fort de cet octet (‘frv’ ou ‘orv’ respectivement) est ‘0’.

Le deuxième bit de poids fort (‘ftv’ ou ‘otv’ respectivement) indique que les six bits de poids faible contiennent une valeur de compteur valide (sauf si une adresse de groupe est utilisée).

6.3.5.5 Codage de l’en-tête de réseau

6.3.5.5.1 Paquets sortants et entrants

La Network_Address finale et la Network_Address d’origine d'un paquet sortant sont utilisées pour générer les champs Link_Header et Network_Header sur le MVB (Topo_Counter n'est pas utilisé), comme illustré à la Figure 125.



Légende

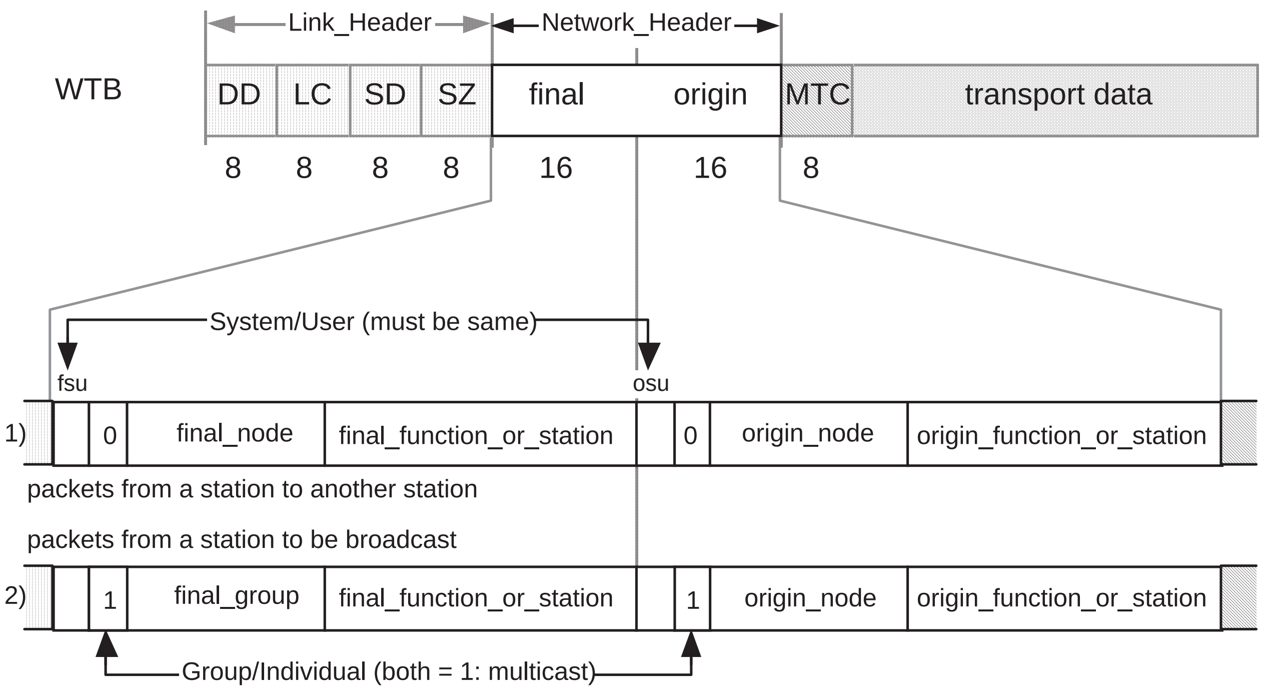
Anglais	Français
transport layer	couche transport
outbound packet	paquet sortant
final network address	adresse de réseau finale
origin network address	adresse de réseau d'origine
final	final
origin	origine
transport data	données de transport

Figure 125 – Génération des adresses dans un paquet sortant

6.3.5.5.2 Codage des en-têtes de réseau sur le bus de train

L'en-tête de réseau sur le bus de train doit être codé selon la spécification suivante (voir la Figure 126 pour le WTB):

```
Network_Header::= RECORD
  {
  fsu                   ENUM1
    {
    USER                (0)                  -- adresse d'utilisateur
                                                (fonction) est utilisée
    SYSTEM              (1)                  -- adresse de système
                                                (station) est utilisée
    },
  fgi                   UNSIGNED1,           -- 1 si groupe, 0 si
                                                individuel
  ONE_OF [fgi]
    {
    [0] final_node      UNSIGNED6,
    [1] final_group     UNSIGNED6
    },
  ONE_OF [fsu]
    {
    [USER] final_function,                   UNSIGNED8,
    [SYSTEM] final_station                   UNSIGNED8
    }
  osu                   ENUM1,               -- identique à 'fsu', si
                                                différent: réservé
  ogi                   UNSIGNED1,           -- identique à 'fgi' (1=
                                                distribution)
  origin_node           UNSIGNED6,           -- l'origine est toujours un
                                                noeud individuel
  ONE_OF [snu]
    {
    [USER] origin_function,  UNSIGNED8,      -- si adresse utilisateur
    [SYSTEM] origin_station  UNSIGNED8       -- si adresse système
    }
  }
```



Légende

Anglais	Français
final	final
origin	origine
transport data	données de transport
system/User (shall be same)	système/utilisateur (doit être identique)
packets from a station to another station	paquets d'une station à une autre
packets from a station to be broadcast	paquets d'une station à diffuser
group/individual (both = 1: multicast)	groupe/individu (les deux = 1: diffusion)

Figure 126 – Codage des adresses réseau sur le bus de train

Origin_Node et Final_Node (ou Group) doivent être définis (<> AM_UNKNOWN).

Final_Address et Origin_Address doivent utiliser le même type d'adressage (système ou utilisateur), la même valeur doit être attribuée aux bits 'snu'.

Origin_Address doit être une Node_Address individuelle.

Si l'adresse finale spécifie un Groupe (diffusion), la valeur 1 doit être attribuée aux bits 'fgi' et 'ogi'.

6.3.5.6 Acheminement dans la Couche Réseau

6.3.5.6.1 Situations

Les neuf situations figurant dans le Tableau 21 définissent le comportement d'une couche réseau généralisée reliée à:

- une couche transport;
- une ou plusieurs couches de liaison de réseau de rame;
- une couche de liaison de bus de train.

Tableau 21 – Situations d'acheminement

Situation	Depuis	Vers	Remarques
1	cette couche transport	cette couche transport	raccourci à l'intérieur du même dispositif
2	cette couche transport	n'importe quel réseaux de rame	il peut exister plusieurs réseaux de rame
3	cette couche transport	le bus de train	la station est le Nœud
4	n'importe quel réseaux de rame	cette couche transport	il peut exister plusieurs réseaux de rame
5	n'importe quel réseaux de rame	Un autre réseau de rame	fonction d'acheminement (routeur)
6	n'importe quel réseaux de rame	le bus de train	fonction d'acheminement (routeur)
7	le bus de train	cette couche transport	la station est le Nœud
8	le bus de train	tous les Réseaux de Rame	fonction d'acheminement (routeur)
9	le bus de train	le bus de train	non prévu

Une station terminale reliée au réseau de rame ne rencontre que les situations 1, 2 et 4.

Une station terminale reliée au bus de train ne rencontre que les situations 1, 3 et 7.

Une station d'acheminement reliée à deux réseaux de rame ne rencontre que les situations 1, 2, 4 et 5.

6.3.5.6.2 Vérification du chemin de retour

La couche réseau doit assurer que tous les paquets appartenant à un Call_Message et à son Reply_Message correspondant empruntent le même chemin.

La couche réseau doit vérifier que pour chaque paquet qu'elle reçoit qu'elle peut renvoyer un paquet à l'Origin_Address, , sinon la couche réseau ne doit pas faire suivre le paquet.

Pour cela, il est supposé que pour un paquet reçu, l'Origin_Address est remplacée par la Final_Address et l'adresse source est remplacée par l'adresse destination, et en vérifiant que dans ce cas, Next_Station est définie.

Lors de l'initialisation, 'next_station' peut être indéfinie, en particulier lorsque la station appelante ne comporte aucune entrée dans le répertoire de stations.

Si un message système est reçu d'une station qui ne figure pas dans son répertoire de stations, la couche réseau doit supposer que la station d'origine dans l'Origin_Address détient la Link_Address correspondant au dispositif émetteur, puis la rentrer dans le répertoire de stations. Un message utilisateur reçu dans ces conditions est rejeté.

NOTE Si nécessaire, cette entrée implicite peut être corrigée par la suite par des messages de gestion.

6.3.5.6.3 Cohérence de la topographie

Pour assurer une protection contre les changements de configuration du bus de train pendant un transfert de message, les paquets transmis vers ou depuis un nœud sur le réseau de rame contiennent le Topo_Counter.

La variable 'this_topo' de la couche réseau doit conserver la valeur la plus récente du Topo_Counter.

NOTE 1 La valeur de 'this_topo' est envoyée à la couche réseau par le service AMI 'am_set_current_tc'.

NOTE 2 Si la station est un Nœud, 'this_topo' est incrémenté lors d'une inauguration.

NOTE 3 Lors d'un appel, une Application indique la valeur de 'my_topo'. Si l'application ne connaît pas la Topographie ou ne tient pas compte de la cohérence, elle met my_topo = AM_ANY_TOPO.

La couche transport envoie 'my_topo' à la couche réseau comme partie intégrante de la Network_Address.

La couche réseau sur un Nœud doit comparer 'this_topo' à la valeur 'packet_topo' trouvée dans les paquets entrants.

Lorsqu'un Nœud reçoit un paquet de son réseau de rame:

- où le packet_topo de l'Origin_Address est égal à son this_topo ou à AM_ANY_TOPO, le Nœud doit insérer sa propre Node_Address (this_node) dans l'Origin_Address et envoyer le paquet sur le bus de train;
- où le packet_topo dans son Origin_Address est différent de this_topo, et de AM_ANY_TOPO, le Nœud doit annuler le transfert (voir 6.3.5.6.4).

Lorsqu'un Nœud reçoit un paquet d'un autre Nœud sur le bus de train:

- un paquet dirigé vers l'une de ses stations ou fonctions, il doit insérer 'this_topo' dans la Final_Address comme packet_topo;
- la station finale doit s'assurer que le packet_topo envoyé par le Nœud est identique à this_topo, sinon la station finale doit annuler le transfert (voir 6.3.5.6.4).

6.3.5.6.4 Annulation

Pour annuler un transfert, la couche réseau doit envoyer le paquet suspect à sa couche transport, qui envoie une Disconnect_Request avec une raison AM_INAUG_ERR à l'origine du paquet.

Lors de l'inauguration, un nœud doit répondre à tout paquet venant du réseau de rame et s'adressant à une autre rame par une Disconnect_Request avec une raison AM_INAUG_ERR.

Lorsqu'un dispositif reçoit une Disconnect_Request avec une raison AM_INAUG_ERR, il doit annuler toutes les connexions en cours sur le bus de train.

NOTE Une couche réseau sans connexion n'étant pas supposée agir sur le protocole, elle délègue cette tâche à la couche transport. Cela ne peut se produire qu'en cas d'erreur d'inauguration.

6.3.5.6.5 Algorithmes d'acheminement

Un Routeur doit faire suivre un paquet selon les dispositions suivantes:

- pour les paquets sortants dans le Tableau 22,
- pour les paquets entrants et en transit provenant d'un réseau de rame dans le Tableau 23, et
- pour les paquets entrants et en transit provenant d'un bus de train dans le Tableau 24.

Tableau 22 – Acheminement des paquets en provenance de la couche transport

Depuis	Vers	Condition
cette couche transport	Quelconque	La couche transport communique next_station et Topo_Counter à la couche réseau de la Network_Address La couche transport ne vérifie pas le Topo_Counter des paquets sortants.
(1)	Couche de transport	En point à point: impossible, puisque la Couche session contourne le réseau lorsque le partenaire réside sur la même station. En distribution: un paquet de diffusion est transmis par cette station et peut être adressé à une fonction résidant sur cette station comme le résultat d'un rebouclage en provenance de la couche de liaison, lorsque: (member AND (fundi(function_id) = this_station)
(2)	Couche de liaison du réseau de rame	(final_node = AM_SAME_NODE) AND (next_station <> AM_SAME_STATION) AND (next_station <> this_station) AND (Link_Address of next_station = defined)
(3)	Couche de liaison du bus de train	(next_station = AM_SAME_STATION) OR ((next_station = this_station) AND (final_node <> AM_SAME_NODE)) Cette situation ne peut exister que sur un Nœud. La couche transport indique de faire suvre ce paquet sur le bus de train en spécifiant: (next_station = AM_SAME_STATION). La couche transport est supposée vérifier l'existence d'une connexion de bus de train (pas d'inauguration en cours, par exemple).

Tableau 23 – Acheminement des paquets en provenance d'un réseau de rame

Depuis	Vers	Condition

Réseau de rame	Quelconque	La couche réseau détermine next_station pour tous les paquets en analysant l'adresse de réseau finale:
		`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)`

4	couche transport	Tous les cas qui ne sont pas couverts par les autres conditions. Le paquet n'est pas envoyé au bus de train, aucune station ne le reçoit. La couche réseau communique la Link_Address du dispositif émetteur à la couche transport dans next_station. La couche transport envoie une Disconnect_Request si le partenaire n'existe pas.

5	un autre réseau de rame	`( (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))`
		Même si un paquet n'est pas envoyé sur le bus de train, sa topographie est vérifiée si (final_node <> AM_SAME_NODE).

6	bus de train	`((final_node <> this_node) AND (final_node <> AM_SAME_NODE)`
		`AND`
		`((packet_topo = this_topo) OR (packet_topo = AM_ANY_TOPO)))`
		`OR`
		`member.`
		Un paquet est envoyé sur le bus de train uniquement si packet_topo est correct ou si la distribution est utilisée.

Tableau 24 – Acheminement des paquets en provenance du bus de train

Depuis	Vers	Condition
Bus de train		La différence principale par rapport aux paquets en provenance d'un réseau de rame est le traitement du Topo_Counter. Dans les paquets en provenance du bus de train, le Nœud introduit son this_topo à la place du final_node pour que la station finale puisse le vérifier.

7	couche transport	`((final_node = this_node) OR (member))`
		`AND`
		`((final_station = this_station) OR`
		`(final_station = AM_SAME_STATION) OR`
		`(final_station = AM_UNKNOWN))`

		Le paquet est destiné à ce Nœud et aucune autre station ne l'accepterait. La couche transport détermine si le partenaire existe

8	réseau de rame	`((final_node = this_node) OR (member))`
		`AND`
		`((final_station <> this_station) AND`
		`(final_station <> AM_SAME_STATION) AND`
		`(final_station <> AM_UNKNOWN))`

		Une station est associée à ce Nœud qui accepterait le paquet.
9	bus de train	Cette situation n'est pas prévue.

6.3.5.7 Interface de couche réseau

L'interface de couche réseau avec la couche transport n'est pas spécifiée et n'est pas ouverte.

L'application peut établir et consulter les répertoires qui font en principe partie de la couche réseau via l'AMI.

6.3.6 Couche de transport de messages

6.3.6.1 Objet

Le présent paragraphe définit deux protocoles de transport:

- le Protocole de Transport de messages (MTP) utilisé pour la communication point à point;
- le Protocole de Distribution (MCP) (en option) utilisé pour la distribution.

6.3.6.2 Protocole de Transport de messages (MTP)

La couche transport transfère un message d'un Producteur vers un Consommateur et fournit les services suivants:

a) la segmentation de longs messages en paquets de taille fixe pour leur transmission;
b) le contrôle de flux et la correction d'erreurs de bout en bout à l'aide d'un protocole à fenêtre glissante;
c) l'annulation.

Le MTP ouvre une connexion unidirectionnelle pendant la durée d'un message.

Le MTP prend en charge le transport simultané de plusieurs messages entre deux processus application.

Le MTP prend en charge le transport simultané de plusieurs messages par la même Application. Chaque connexion se distingue par un paramètre de référence de Couche session, session_ref.

Toutefois, le MTP ne prend en charge qu'une seule connexion simultanée et unidirectionnelle entre les deux mêmes Producteur et Consommateur.

NOTE Les termes 'Producteur' et Consommateur' désignent l'émetteur et le destinataire d'un message au niveau de la couche transport. Les termes 'origine' et 'final' désignent les partenaires finaux sur le réseau. Les termes 'source' et 'destination' désignent la Device_Address des partenaires sur le même bus.

Le Messager exécute les protocoles dans chaque Station en parallèle avec les Applications, en tant que processus séparé.

Un Appelant envoie un Call_Message au Messager par l'intermédiaire de l'ALI.

La Session_Layer du messager ouvre la connexion (et la ferme par la suite).

La Transport_Layer divise le message en une séquence de paquets de données suffisament petits, de manière à pouvoir les intégrer aux trames de bus et les envoyer à la Network_Layer.

La Network_Layer consulte ses répertoires de fonctions et de stations afin de traduire les adresses des paquets et de transmettre les paquets à la Link_Layer.

Le Messager éloigné signale l'arrivée d'un message complet au Répondeur, qui peut alors répondre par un Reply_Message dans la direction opposée.

Le Protocole de Transfert de Messages exécuté par la Transport_Layer du Producteur et du Consommateur procède à un contrôle de flux et une correction des erreurs afin d'éviter toute perte ou duplication de paquets.

Si les deux Applications résident sur la même Station, la Session_Layer contourne le réseau et évite la segmentation. Par exemple, les Applications Utilisateur peuvent nécessiter les services de l'Agent local en lui envoyant un message sans accéder au réseau.

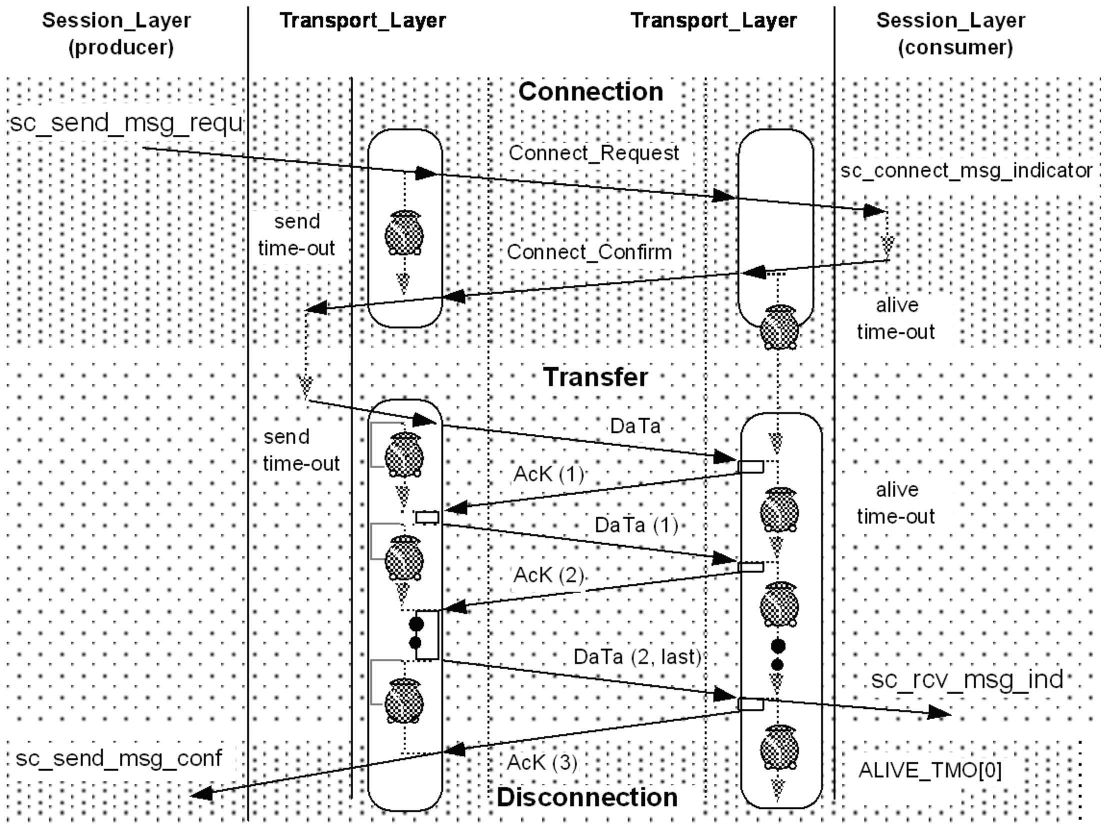
Le Messager de chaque Station exécute le protocole de transmission. Le Messager au niveau du Producteur divise le message en paquets de données, que le Messager au niveau du Consommateur reconnaît au moyen de paquets de contrôle.

6.3.6.3 Échange de Paquets

La transmission d'un message doit être divisée en trois phases:

a) l'établissement de la connexion;
b) le transfert de données avec accusé de réception;
c) la déconnexion.

EXEMPLE La Figure 127 illustre un échange simple de paquets avec une taille de fenêtre de 1.



Légende

Anglais	Français
(producer)	(producteur)
(consumer)	(consommateur)
Connection	Connexion
send time-out	envoyer temporisation
alive time-out	temporisation en cours
Transfer	Transfert
Disconnection	Déconnexion

Figure 127 – Echange de paquets de transport

6.3.6.3.1 Etablissement de la connexion

Le Producteur demande au Consommateur d'accepter un message en envoyant un paquet Connect_Request. Ce paquet contient la longueur totale du message, la taille des paquets, la taille de la fenêtre, ainsi que la Référence de Connexion fournie par la couche transport qui identifie de manière unique la Connect_Request si celle-ci doit être répétée.

Le Consommateur répond à un paquet Disconnect_Request soit par une Connect_Confirm s'il accepte le message, soit par un paquet Disconnect_Request s'il le rejette.

La taille de la fenêtre est négociée lors de l'établissement de la connexion: le Producteur propose un crédit et le Consommateur répond avec le crédit accepté, qui ne peut dépasser le crédit proposé.

Le même principe s'applique pour la taille du paquet. La taille du paquet Connect_Request est la plus petite possible sur le réseau, ce qui est déterminé par les 256 bits qu'un bus MVB est capable d'envoyer.

Le crédit et la taille du paquet que le Consommateur renvoie s'appliquent à tous les paquets suivants du même message.

Le Producteur peut insérer jusqu'à 14 octets de données dans le paquet Connect_Request.

Un message de 14 octets au maximum peut être intégralement envoyé au Consommateur sans paquet de données. Le paquet Connect_Confirm accuse alors réception de l'ensemble du message.

Si la Connect_Request n'est pas confirmée ou annulée, la procédure doit tenter de retransmettre le message au moins trois fois avant de déconnecter.

6.3.6.3.2 Transfert de données avec accusé de réception

Le Producteur envoie les paquets de données individuels du message au Consommateur. Il peut envoyer les paquets AcpCredit (Crédit Accepté) sans recevoir un accusé de réception pour eux.

Le Consommateur doit accuser réception des paquets de données de façon positive par un paquet d'accusé de réception en indiquant le numéro du paquet suivant attendu.

Le Consommateur peut grouper les accusés de réception, c'est-à-dire accuser réception de plusieurs paquets en même temps en n'accusant réception que du paquet portant le numéro de séquence le plus élevé.

Le Producteur doit être capable de traiter les accusés de réception groupés.

Lorsqu'il n'est plus accusé réception des paquets, la couche transport doit les retransmettre au maximum trois fois avant de se déconnecter.

Le Consommateur peut informer le Producteur qu'il a reçu un paquet hors séquence. Dans ce cas, il doit envoyer un paquet Negative_Acknowledgement en indiquant à partir de quel paquet il demande une retransmission.

6.3.6.3.3 Déconnexion

La déconnexion est implicite si le transfert n'est pas annulé de manière explicite. Cependant, la connexion n'est pas coupée tout de suite puisque des paquets tardifs peuvent toujours arriver en cas de perte des accusés de réception.

La déconnexion est explicite si le Producteur ou le Consommateur envoie un paquet Disconnect_Request.

La partie qui reçoit une Disconnect_Request (sauf en réponse à une Connect_Request ou à une Disconnect_Request avec la raison AM_REM_CANC) répond par un paquet Disconnect_Confirm.

Faisant exception à cette règle, une station d'acheminement peut envoyer une Disconnect_Request si elle ne peut faire suivre les paquets correctement (suite à un changement de composition, par exemple).

NOTE Étant donné qu'une couche réseau n'est pas censée agir sur le protocole, ce paquet est envoyé à la couche transport du routeur, qui envoie la Disconnect_Request.

6.3.6.4 Référence de connexion

Chaque Connect_Request doit être identifiée par un paramètre Connection_Reference.

La Connection_Reference doit être un nombre à 16 bits initialisé à une valeur arbitraire au démarrage (pris de l'horloge en temps réel, par exemple).

La couche transport doit incrémenter la Connection_Reference de 1 à chaque nouvelle connexion sortante de cette station.

La Connection_Reference doit être utilisée pour chaque paire appel/réponse de l'application appelante.

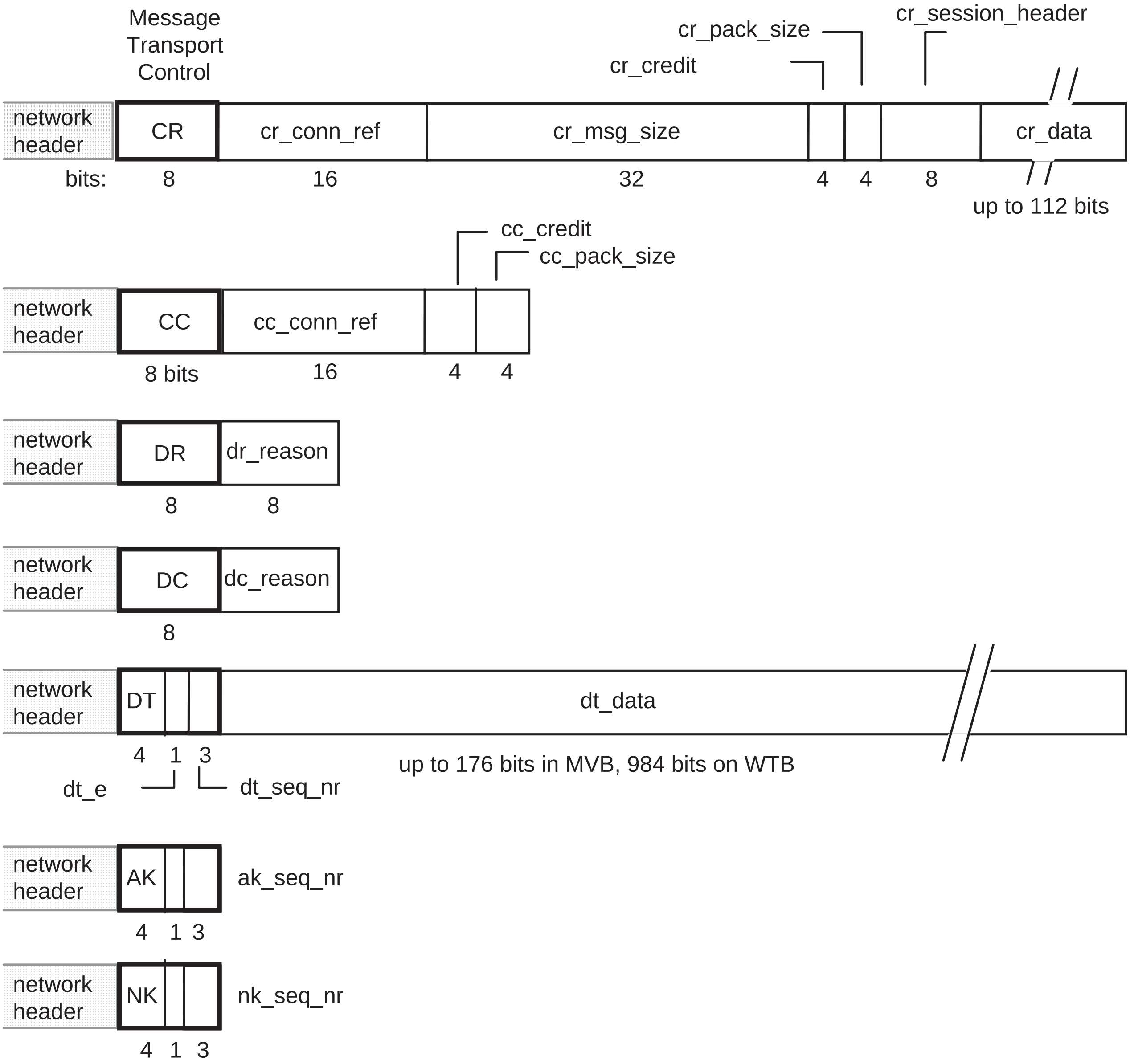
NOTE 1 La Connection_Reference fait une distinction entre la répétition d'une Connect_Request (en cas d'erreur de transmission) et une nouvelle demande différente.

NOTE 2 La Connection_Reference est appelée conn_ref dans les tableaux de transition d'état.

6.3.6.5 Paquets de la couche transport

6.3.6.5.1 Types de paquets

La couche transport doit fonctionner avec les sept types de paquets illustrés à la Figure 128 et définis dans les paragraphes ci-dessous.



Légende

Anglais	Français
message transport control	contrôle de transport des messages
network header	en-tête de réseau
up to 176 bits in MVB, 984 bits on WTB	jusqu'à 176 bits sur MVB, 984 bits sur WTB

Figure 128 – Format des paquets (corps de la couche transport)

NOTE 1 L'en-tête de réseau est défini dans la spécification de la Couche Réseau.

NOTE 2 L'en-tête de liaison est défini dans la spécification la couche de liaison (il est lié au bus).

6.3.6.5.2 Codage du Contrôle de Transport de messages

Le champ Contrôle de Transport de messages doit être codé selon les dispositions du Tableau 25.

Tableau 25 – Codage du Contrôle de transport de messages

Type de Paquet	Paquet	Abré-viation	Codage							
	Connect_Request	CR	cl	0	0	0	0	0	0	0
Contrôle	Connect_Confirm	CC	cl	1	0	0	0	0	0	1
(point à point)	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	
Diffusion	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		
Informations	Acknowledgement	AK	cl	1	1	0	ak_e	ak_seq_nr		
	Negative Acknowledgement	NK	cl	1	1	1	nk_e	nk_seq_nr		

Le premier bit du champ de contrôle (cl) doit faire la distinction entre un datagramme Appel (1) et Réponse (0).

Le deuxième bit (co) doit faire la distinction entre le Consommateur (1) et le Producteur (0).

La valeur 1 doit être attribuée au bit dt_e (fin de transfert) du dernier paquet d'un message.

Les combinaisons non définies sont réservées.

NOTE CL fait partie de la couche session, mais est également utilisé par la couche transport.

6.3.6.5.3 Codage de Am_Result

Le résultat d'un transfert doit être codé dans un paquet par un champ de type Am_Result:

```
Am_Result::=           ENUM8
  {
  AM_OK                (0)        -- terminé avec succès
  AM_FAILURE           (1)        -- erreur non spécifiée
  AM_BUS_ERR           (2)        -- pas de transmission possible sur
                                            le bus
  AM_REM_CONN_OVF      (3)        -- trop de connexions entrantes
  AM_CONN_TMO_ERR      (4)        -- pas de réponse à Connect_Request
  AM_SEND_TMO_ERR      (5)        -- temporisation SEND_TMO terminée
  AM_REPLY_TMO_ERR     (6)        -- pas de réponse reçue
  AM_ALIVE_TMO_ERR     (7)        -- temporisation ALIVE_TMO terminée
  AM_NO_LOC_MEM_ERR    (8)        -- pas assez de mémoire ou d'horloges
  AM_NO_REM_MEM_ERR    (9)        -- plus de mémoire ou d'horloges chez
                                            le partenaire
  AM_REM_CANC_ERR      (10)       -- annulé par le partenaire
  AM_ALREADY_USED      (11)       -- même operation déjà effectuée
  AM_ADDR_FMT_ERR      (12)       -- erreur de format d'adresse
  AM_NO_REPLY_EXP_ERR  (13)       -- type de réponse non attendu
  AM_NR_OF_CALLS_OVF   (14)       -- trop d'appels demandés
  AM_REPLY_LEN_OVF     (15)       -- Reply_Message trop long
  AM_DUPL_LINK_ERR     (16)       -- erreur de conversation dupliquée
  AM_MY_DEV_UNKNOWN_ERR           (17)      -- mon dispositif inconnu
  AM_NO_READY_INST_ERR            (18)      -- pas de répondeur prêt
  AM_NR_OF_INST_OVF    (19)       -- trop de répondeurs
  AM_CALL_LEN_OVF      (20)       -- Call_Message trop long
  AM_UNKNOWN_DEST_ERR  (21)       -- dispositif du partenaire inconnu
  AM_INAUG_ERR         (22)       -- inauguration du train faite
  AM_TRY_LATER_ERR     (23)       -- (utilisation interne seulement)
  AM_FIN_NOT_REG_ERR   (24)       -- adresse finale pas enregistrée
  AM_GW_FIN_NOT_REG_ERR           (25)      -- adresse finale pas
                                            enregistrée dans le router
  AM_GW_ORI_REG_ERR    (26)       -- adresse origine pas enregistrée
                                            dans le router
  AM_MAX_ERR           (31)       -- code système le plus haut.
  }                               -- codes utilisateurs supérieurs à 31
```

NOTE Le type énuméré Am_Result correspond au paramètre AM_RESULT qui est retourné par les procédures de la couche application, mais également par les procédures de la couche session et de la couche transport. AM_RESULT est le type d'un paramètre exprimé dans la syntaxe du langage C, alors que Am_Result définit le codage de ce type pour la transmission.

6.3.6.5.4 Codage de paquet (voir du Tableau 26 au Tableau 36)

Les paquets doivent être formatés de la manière suivante:

```
Message_Packet::= RECORD
  {
  call_not_reply      ENUM1              -- le premier bit distingue
                                            l'appel de la réponse
    {
    REPLY_MSG  (0),                      -- message de réponse
    CALL_MSG   (1)                       -- message d'appel
  consumer_not_producer                  ENUM1   -- le second bit
                                            distingue le consommateur
                                            du producteur
    {
    CONSUMER (0),                        -- en provenance du
                                            consommateur
    PRODUCER (1)                         -- en provenance du producteur
    },
  packet_kind         ENUM2              -- les 3e et 4e bits
                                            distinguent le type de
                                            paquet
    {
    CONTROL (0)                          -- message de contrôle
    DATA    (1)                          -- transfert de données
    ACK     (2)                          -- accusé de réception positif
    NAK     (3)                          -- accusé de réception négatif
  ONE_OF [packet_type]
    {
    [CONTROL]         Control_Packet     -- contrôle
    [DATA]            Data_Packet,       -- transfert de données
    [ACK]             Ack_Packet,        -- accusé de réception positif
    [NAK]             Nak_Packet         -- accusé de réception négatif
    }
  }
```

```
Control_Packet::= RECORD
  {
  command::==         ENUM4                -- les quatre derniers bits de
                                              l'octet MTC indiquent la
                                              commande
    {
    CR  (0),                               -- Connect_Request
    CC  (1),                               -- Connect_Confirm
    DR  (2),                               -- Disconnect_Request
    DC  (3)                                -- Disconnect_Confirm
    BC1 (8)                                -- Broadcast_Request 1er essai
                                              (option)
    BC2 (9)                                -- Broadcast_Request 2e essai
                                              (option)
    BC3 (10)                               -- Broadcast_Request 3e essai
                                              (option)
    BS  (12)                               -- Broadcast_Stop  (option)
    BR  (13)                               -- Broadcast_Repeat (option)
    BD  (14)                               -- Broadcast_Data   (option)
    },
  ONE_OF [command]                         -- dépend de MTC
    {
    [CR]              Connect_Request,
    [CC]              Connect_Confirm,
    [DR]              Disconnect_Request,
    [DC]              Disconnect_Confirm,
    [BC1]             Broadcast_Connect,     -- voir 6.3.7.5
    [BC2]             Broadcast_Connect,     -- voir 6.3.7.5
    [BC3]             Broadcast_Connect,     -- voir 6.3.7.5
    [BR]              Broadcast_Repeat,      -- voir 6.3.7.5
    [BS]              Broadcast_Stop,        -- voir 6.3.7.5
    [BD]              Broadcast_Data         -- voir 6.3.7.5
    }
  }
```

Tableau 26 – Connect_Request

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>cl</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="8">cr_conn_ref</td></tr>
<tr><td colspan="8">cr_msg_size</td></tr>
<tr><td colspan="4">cr_credit</td><td colspan="4">cr_pack_size</td></tr>
<tr><td colspan="8">cr_session_header</td></tr>
<tr><td colspan="8">cr_data

(jusqu'à 14 octets)</td></tr>
</table>

```
Connect_Request::= RECORD
  {
  cr_conn_ref          UNSIGNED16          -- référence de connexion de
                                              16 bits
  cr_msg_size          UNSIGNED32          -- taille totale du message en
                                              octets, maximum 4 Go
  cr_credit            UNSIGNED4           -- Crédit proposé:
                                              ‘0000’B = 0 (stop
                                              communication)
                                              ‘0111’B = 7 (maximum)
  cr_pack_size         ENUM4               -- taille de paquet proposée,
    {
    MVB_PACKET (0),                        -- Transport_Data =
                                              22 octets/paquet (MVB)
    WTB_PACKET (1)                         -- Transport_Data = 123
                                              octets/paquet (WTB)
                                              autres valeurs réservées.
    },
  cr_session_header    Am_Result           -- AM_OK dans un appel
                                              demandant une réponse
  cr_data              ARRAY [14] OF WORD8    -- jusqu’à 14 octets de
                                              Transport_Data
  }
```

Tableau 27 – Connect_Confirm

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>cl</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">cc_conn_ref</td></tr>
<tr><td colspan="4">cc_credit</td><td colspan="4">cc_pack_size</td></tr>
</table>

```
Connect_Confirm::= RECORD
  {
  cc_conn_ref          UNSIGNED16          -- même valeur que celle reçue
                                              dans Connect_Request.
  cc_credit            UNSIGNED4           -- crédit accepté:
                                              ‘0000’B = 0 (stop
                                              communication)
                                              ‘0111’B = 7 (maximum)
  cc_pack_size         ENUM4               -- taille de paquet acceptée
    {
    MVB_PACKET (0),                        -- Transport_Data_Size = 22
                                              octets/paquet (MVB)
    WTB_PACKET (1)                         -- Transport_Data_Size = 123
                                              octets/paquet (WTB)
                                           -- autres valeurs réservées.
    }
  }
```

Tableau 28 – Disconnect_Request

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

```
Disconnect_Request::= RECORD
  {
  dr_reason            Am_Result           -- raison de la déconnexion
  }
```

Tableau 29 – Disconnect_Confirm

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

```
Disconnect_Confirm::= RECORD
  {
  dc_reason            Am_Result           -- la raison est AM_OK si la
                                              déconnexion est réussie.
  }
```

Tableau 30 – Data_Packet

7	6	5	4	3	2	1	0
cl	0	0	1	dt_e	dt_seq_nr		
dt_data							
(jusqu’à Transport_Data_Size octets)							

```
Data_Packet::= RECORD                     -- débute avec les derniers
                                             quatre bits de l'octet MTC
  {
  dt_e                BOOLEAN1            -- 1 = fin du message (pas de
                                             la session)
  dt_seq_nr           UNSIGNED3           -- numéro de séquence de ce
                                             paquet vu par le
                                             Producteur.
  dt_data             ARRAY [transport_data_size] OF WORD8
  }                                       -- sauf peut-être dans le
                                             dernier paquet du message.
```

Tableau 31 – Ack_Packet

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

```
Ack_Packet::= RECORD                      -- derniers 4 bits de MTC
  {
  ak_e                ENUM1 (=0)          -- toujours FALSE (0)
  ak_seq_nr           UNSIGNED3           -- numéro du prochain paquet
                                             attendu par le Consommateur
  }
```

Tableau 32 – Nak_Packet

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

```
Nak_Packet::= RECORD
  {                                       -- quatre derniers bits de MTC
  nk_eot              BOOLEAN1            -- toujours FALSE (0)
  nk_seq_nr           UNSIGNED3           -- numéro du prochain paquet
                                             attendu par le Consommateur
  }
```

Les paquets suivants ne sont utilisés qu'avec la diffusion (en option):

Tableau 33 – Broadcast_Connect (BC1, BC2, BC3)

7	6	5	4	3	2	1	0
1	0	0	0	1	0	bc_rept	
bc_run_nr							
bc_msg_size							
bc_data							
(jusqu'à 18 octets)							

```
Broadcast_Connect::= RECORD                -- spécifié ici pour 6.3.7.5
  {                                        -- BC1, BC2 et BC3 sont
                                              distinguées par bc_rept
  bc_run_nr          UNSIGNED16            -- Connection_Reference
                                              identifiant ce message
  bc_msg_size        UNSIGNED16            -- taille totale du message en
                                              octets
  bc_data            ARRAY [18] OF WORD8   -- jusqu'à 18 octets de
                                              données utilisateur
  }
```

Tableau 34 – Broadcast_Data

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

```
Broadcast_Data::= RECORD                   -- spécifié ici pour 6.3.7.5
  {
  bd_run_nr          UNSIGNED16            -- Connection_Reference
                                              identifiant ce message
  bd_offset          UNSIGNED16            -- position en octets des
                                              données bd_data du message
  bd_data            ARRAY [18] OF WORD8   -- jusqu'à 18 octets de
                                              données utilisateur
  }
```

Tableau 35 – Broadcast_Repeat

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

```
Broadcast_Repeat::= RECORD                 -- spécifié ici pour 6.3.7.5
  {
  br_run_nr          UNSIGNED16            -- Connection_Reference
                                              identifiant ce message
  br_offset          UNSIGNED16            -- position en octets des
                                              données bd_data manquantes
  }
```

Tableau 36 – Broadcast_Stop (BSC, BSO)

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

```
Broadcast_Stop::= RECORD                 -- spécifié ici pour 6.3.7.5
  {
  bs_run_nr            UNSIGNED16        -- Connection_Reference
                                            identifiant ce message
  }
```

6.3.6.6 Diagrammes de transition d'état

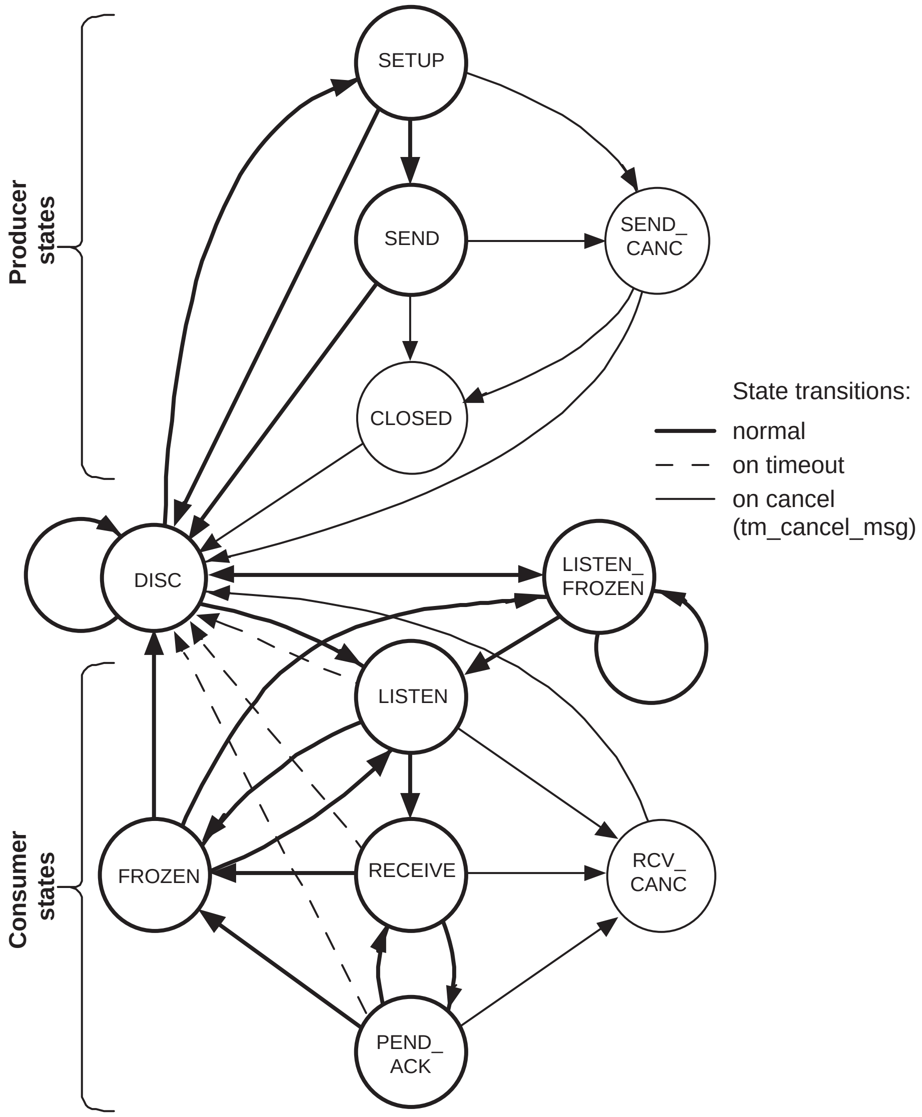
6.3.6.6.1 Etats

Le Protocole de Transport de messages doit mettre en œuvre une machine d'état MTP avec les états donnés dans le Tableau 37 pour chaque Producteur ou Consommateur.

Tableau 37 – Etats MTP

Nom d'état	Concerne	Brève description
DISC	Producteur, Consommateur	déconnecté
SETUP	Producteur	en attente d'un paquet CC
SEND	Producteur	ouvert pour envoyer un message
SEND_CANC	Producteur	paquet DR envoyé, en attente d'un paquet DC ou DR
CLOSED	Producteur	paquet DR reçu
LISTEN	Consommateur	en attente d'un accusé de réception du paquet CC
RCV_CANC	Consommateur	paquet DR envoyé, en attente d'un paquet DC ou DR
RECEIVE	Consommateur	ouvert pour la réception d'un message et la réception de tous les paquets reçus a été accusée
PEND_ACK	Consommateur	ouvert pour la réception d'un message et la réception de tous les paquets reçus n'a pas été accusée
FROZEN	Consommateur	message complet avec le paquet DT (EOT) reçu
LISTEN_FROZEN	Consommateur	message complet avec le paquet CR reçu

Les états et les transitions d'une instance de la couche transport (Producteur ou Consommateur) sont représentés à la Figure 129 et décrits ci-dessous.



Légende

Anglais	Français
producer states	états du producteur
consumer states	états du consommateur
state transitions	transitions d'état
normal	normal
on timeout	en cours de temporisation
on cancel	en cours d'annulation

Figure 129 – Diagramme de transition d'état du MTP

6.3.6.6.2 Événements entrants

Les événements entrants détaillés dans le Tableau 38, qui sont générés par le réseau, l'utilisateur ou les temporisations, peuvent provoquer les transitions d'un état MTP vers un autre.

Tableau 38 – Evénements MTP entrants

Événement	Interface	Description brève des événements
tm_send_req	de l'utilisateur	demande d'envoi d'un message, l'utilisateur envoie un tampon de message vers le messager
tm_cancel_req	de l'utilisateur	annule un transfert de message entrant ou sortant le messager retourne le tampon de message à l'utilisateur si l'annulation réussit
rcv_CR	du réseau	paquet CR reçu
rcv_CC	du réseau	paquet CC reçu
rcv_DT	du réseau	paquet DT reçu
rcv_AK	du réseau	paquet AK reçu
rcv_NK	du réseau	paquet NK reçu
rcv_DR	du réseau	paquet DR reçu
rcv_DC	du réseau	paquet DC reçu
TMO	Interne	une temporisation s'est écoulée (il n'y en a qu'une seule en cours à la fois)

6.3.6.6.3 Événements sortants

Les événements sortants détaillés dans le Tableau 39 qui sont générés par la machine d'état MTP, peuvent provoquer des transitions d'un état MTP à un autre dans une autre machine MTP.

Tableau 39 – Evénements MTP sortants

Événement	Interface	Description brève des événements
tm_connect_ind	indication pour l'utilisateur	l'utilisateur doit accepter ou rejeter la Connect_Request entrante; l'utilisateur envoie un tampon de message au messager s'il accepte le message
tm_receive_ind	indication pour l'utilisateur	message complet reçu ou erreur reçue le messager renvoie le tampon de message à l'utilisateur
tm_send_cnf	confirmation pour l'utilisateur	message complet envoyé ou erreur envoyée le messager renvoie le tampon de message à l'utilisateur
send_CR	au réseau	transmission d'un paquet CR
send_CC	au réseau	transmission d'un paquet CC
send_DT	au réseau	transmission d'un paquet DT
send_AK	au réseau	transmission d'un paquet AK
send_NK	au réseau	transmission d'un paquet NK
send_DR	au réseau	transmission d'un paquet DR
send_DC	au réseau	transmission d'un paquet DC

6.3.6.6.4 Paramètres de contrôle dans les paquets

Les paquets échangés doivent contenir les paramètres de contrôle présentés dans le Tableau 40.

Tableau 40 – Paramètres de contrôle MTP

Événement	Champs	Description brève des champs
send_CR (fields) rcv_CR (fields)	CR_conn_ref, CR_credit, CR_pack_size	référence de connexion crédit proposé code de la taille du paquet proposé
send_CC (fields) rcv_CC (fields)	CC_conn_ref, CC_credit, CC_pack_size	référence de connexion crédit accepté code de la taille du paquet accepté
send_DT (fields) rcv_DT (fields)	DT_seq_nr, DT_eot	numéro du paquet drapeau de fin de transfert de message
send_NK (fields) rcv_NK (fields)	NK_seq_nr	numéro du paquet suivant attendu
send_AK (fields) rcv_AK (fields)	AK_seq_nr	numéro du paquet suivant attendu
send_DR (fields) rcv_DR (fields)	DR_reason	raison de la déconnexion, code d'erreur Am_Result

Les paramètres courants des événements d'envoi (send_xx) sont spécifiés pour les champs de contrôle, tandis que pour les événements de réception (rcv_xx), les noms formels des champs sont cités dans les actions.

6.3.6.6.5 Variables auxiliaires

Le diagramme de transition d'état utilise les variables auxiliaires figurant dans le Tableau 41 pour réduire le nombre d'états. Certaines variables n'existent que chez le Producteur ou chez le Consommateur et appartiennent également à une connexion, tandis que les variables globales sont communes à toutes les connexions.

Tableau 41 – Variables auxiliaires MTP

Nom de la variable	Contexte	Description brève de la variable
my_credit	Général	mon crédit maximal admis
run_nr	Général	Connection_Reference libre suivante
my_pack_size	Général	mon code de taille maximale de paquet admise
cancelled	Producteur, Consommateur	ce message a été annulé par l'utilisateur
credit	Producteur, Consommateur	crédit accepté pour cette connexion
expected	Producteur, Consommateur	limite inférieure de la fenêtre d'envoi (Producteur) ou de réception (Consommateur)
conn_ref	Producteur, Consommateur	Connection_Reference de cette connexion
rep_cnt	Producteur, Consommateur	nombre de répétitions
new_cnt	Consommateur	nombre de paquets sans accusé de réception
next_send	Producteur	numéro du paquet suivant à transmettre
send_not_yet	Producteur	limite supérieure de la fenêtre d'envoi
eot	Producteur	message complet transmis
size	Producteur	taille de paquet acceptée pour cette connexion
error	Producteur	code d’erreur AM_xxx pour compte rendu par send_confirm

6.3.6.6.6 Actions et temporisations

Chaque connexion doit avoir sa propre horloge pour la supervision de la temporisation.

L'horloge doit être exploitée par les actions restart_tmo et reset_tmo et doit générer un événement interne TMO lorsqu'il est écoulé.

Les transitions d'état sont contrôlées par trois temporisations:

- SEND_TMO, la temporisation d'envoi du Producteur,
- ACK_TMO, la temporisation d'accusé de réception du Consommateur, et
- ALIVE_TMO, la temporisation courante du Consommateur.

Si le Producteur ne reçoit aucun paquet d'accusé de réception pour un paquet s'il est envoyé dans SEND_TMO, il doit retransmettre le même paquet.

Si le Producteur ne reçoit aucun paquet d'accusé de réception pour un paquet suite à des tentatives infructueuses d’envoi du message MAX_REP_CNT, il doit se déconnecter.

La valeur maximale du compteur de répétition MAX_REP_CNT doit être trois.

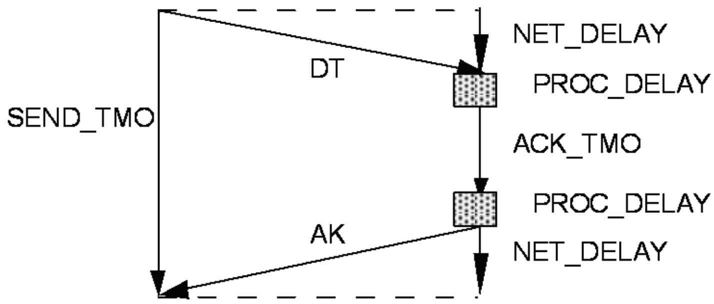
Le Consommateur doit immédiatement accuser réception lorsque le crédit est utilisé ou que le message a été reçu en entier.

La valeur maximale d'un crédit AM_MAX_CREDIT doit être sept.

Sinon, le Consommateur doit retarder l'envoi de l'accusé de réception par la temporisation ACK_TMO, afin d'accuser réception de plusieurs paquets par un seul accusé de réception.

La valeur de la temporisation d'envoi SEND_TMO est un paramètre de configuration.

La temporisation SEND_TMO doit être plus de deux fois la somme du cas le plus défavorable de retard de transmission sur le réseau (NET_DELAY), plus le temps de traitement du paquet dans une station (PROC_DELAY), plus la temporisation d'accusé de réception ACK_TMO (voir la Figure 130).



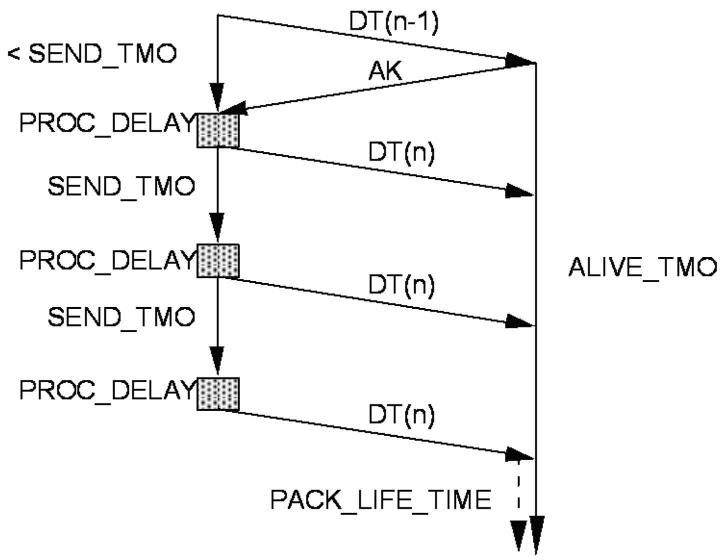
SEND_TMO = 2 × (NET_DELAY + PROC_DELAY) +

Figure 130 – Temporisation SEND_TMO

Le Consommateur d'un paquet doit attendre le paquet suivant (avec le même numéro de séquence ou un numéro différent) dans ALIVE_TMO. S'il ne reçoit pas de paquets pendant cette période, il doit interrompre la conversation.

La valeur de la temporisation courante LIVE_TMO est un paramètre de configuration.

La valeur de la temporisation courante ALIVE_TMO doit être au moins MAX_REP_CNT fois la valeur de la temporisation d'envoi SEND_TMO, plus la durée de vie des paquets PACK_LIFE_TIME dans les files d'attente de liaison (voir la Figure 131).



ALIVE_TMO = MAX_REP_CNT × (SEND_TMO + PROC_DELAY) +

Figure 131 – Temporisation ALIVE_TMO

Le Tableau 42 donne les valeurs par défaut pour la pire des applications.

Tableau 42 – Temporisations MTP (pire des cas)

	NET_ DELAY	PROC_ DELAY	PACK_LIFE_ TIME	ACK_ TMO	SEND_ TMO	ALIVE_ TMO
même bus	0,5 s	64 ms	5,0 s	0,25 s	1,4 s	9,5 s
réseau de rame vers réseau de rame sur le bus de train	9,6 s	64 ms	5,0 s	2.5 s	21,0 s	71,0 s

NOTE Le PACK_LIFE_TIME est spécifié dans la couche de liaison correspondante.

6.3.6.6.7 Actions implicites

Les descriptions d'action dans les transitions d'état suivantes peuvent contenir les actions implicites figurant dans le Tableau 43.

Tableau 43 – Actions Implicites

Référence	Actions implicites
(1)	L'état AM_BUS_ERR est signalé si aucun paquet CR n'a pu être envoyé sur le bus
(2)	opérations avec numéros de paquet modulo 8, comparaisons concernant le retour
(3)	dans ce cas, la valeur TRUE est toujours attribuée à ‘cancelled’

6.3.6.6.8 Actions composées

Les actions présentées dans le Tableau 44 sont exécutées dans plusieurs transitions d'état.

Tableau 44 – Actions Composées

Nom de l'action	Description brève de l'action	État suivant
restart_tmo (xxx_TMO)	arrête le temporisateur et le redémarre avec la temporisation xxx_TMO;	-
reset_tmo	arrête le temporisateur;	-

close_send (error)	IF (NOT cancelled) THEN
	tm_send_cnf (error);
	END;
	reset_tmo;
		DISC

close_rcv (error)	IF (NOT cancelled) THEN
	tm_receive_ind (error);
	END;
	IF (error <> AM_OK) THEN
	reset_tmo;
	ELSE
	restart_tmo (ALIVE_TMO);
	END;
		DISC

update (eot)	eot:= message complet avec le paquet suivant;	

send_data_or_cancel	IF cancelled THEN
	send_DR (AM_REM_CANC_ERR);
	rep_cnt:= 0;
	restart_tmo (SEND_TMO);
	ELSE
	REPEAT
	update (eot);
	send_DT (next_send, eot);
	next_send:= next_send + 1; (2)
	UNTIL (eot OR (next_send = send_not_yet));
	rep_cnt:= 0;
	restart_tmo (SEND_TMO);
	END;
		SEND_CANC
		SEND

6.3.6.6.9 Tableau des événements d'état producteur

Lors d'un événement, le Producteur dans un état courant doit passer à l'état suivant et effectuer l'action spécifiée dans le Tableau 45.

Tableau 45 – Etats et transitions du Producteur

État Actuel	Événement	Action(s)	État suivant (if<>current)
(quelconque)	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
			
			SEND ou
			SEND_CANC

	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);	

Tableau 45 *(suite)*

État Actuel	Événement	Action(s)	État suivant (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;
			SEND ou
			SEND_CANC

		ELSIF (expected = next_send) THEN close_send (AM_OK);	DISC

		ELSE
		REPEAT
		send_DT(expected, eot);
		expected:= expected + 1; (2)
		UNTIL (expected = next_send);
		restart_tmo (SEND_TMO);
		rep_cnt:= rep_cnt + 1;
		END;
			SEND

	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);	

	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
			SEND_CANC

Tableau 45 *(suite)*

État Actuel	Événement	Action(s)	État suivant (if<>current)

		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	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

6.3.6.6.10 Tableau des événements d'état consommateur

Lors d'un événement, le Consommateur dans un état courant doit passer à l'état suivant et effectuer l'action spécifiée dans le Tableau 46.

Tableau 46 – États et transitions du Consommateur

<table>
<tr><th>État Actuel</th><th>Événement</th><th>Action(s)</th><th>État suivant (if<>current)</th></tr>
<tr><td>(any)</td><td>tm_cancel_req</td><td>cancelled:= TRUE;</td><td></td></tr>
<tr><td>DISC</td><td>rcv_DR</td><td>send_DC (dc_reason);</td><td></td></tr>
<tr><td>DISC or
FROZEN</td><td>rcv_CR</td><td rowspan="2">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);
IF (eot) THEN
close_rcv (AM_OK);
ELSE
restart_tmo (ALIVE_TMO);
END;
ELSE
send_DR (err);
END;</td><td rowspan="2">LISTEN_ FROZEN
LISTEN
DISC</td></tr>
<tr><td>LISTEN_ FROZEN</td><td>rcv_CR AND
(CR_conn_ref <> conn_ref)</td></tr>
<tr><td>LISTEN_ FROZEN</td><td>rcv_CR AND
(CR_conn_ref = conn_ref)</td><td>send_CC
(conn_ref, credit,
min (CR_pack_size,
my_pack_size);</td><td></td></tr>
<tr><td>FROZEN OR
LISTEN_ FROZEN</td><td>TMO</td><td></td><td>DISC</td></tr>
</table>

Tableau 46 *(suite)*

État courant	Événement	Action(s)	État suivant (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(dc_reason); 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

Tableau 46 *(fin)*

État courant	Événement	Action(s)	État suivant (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

6.3.6.7 Interface de transport de messages (TMI)

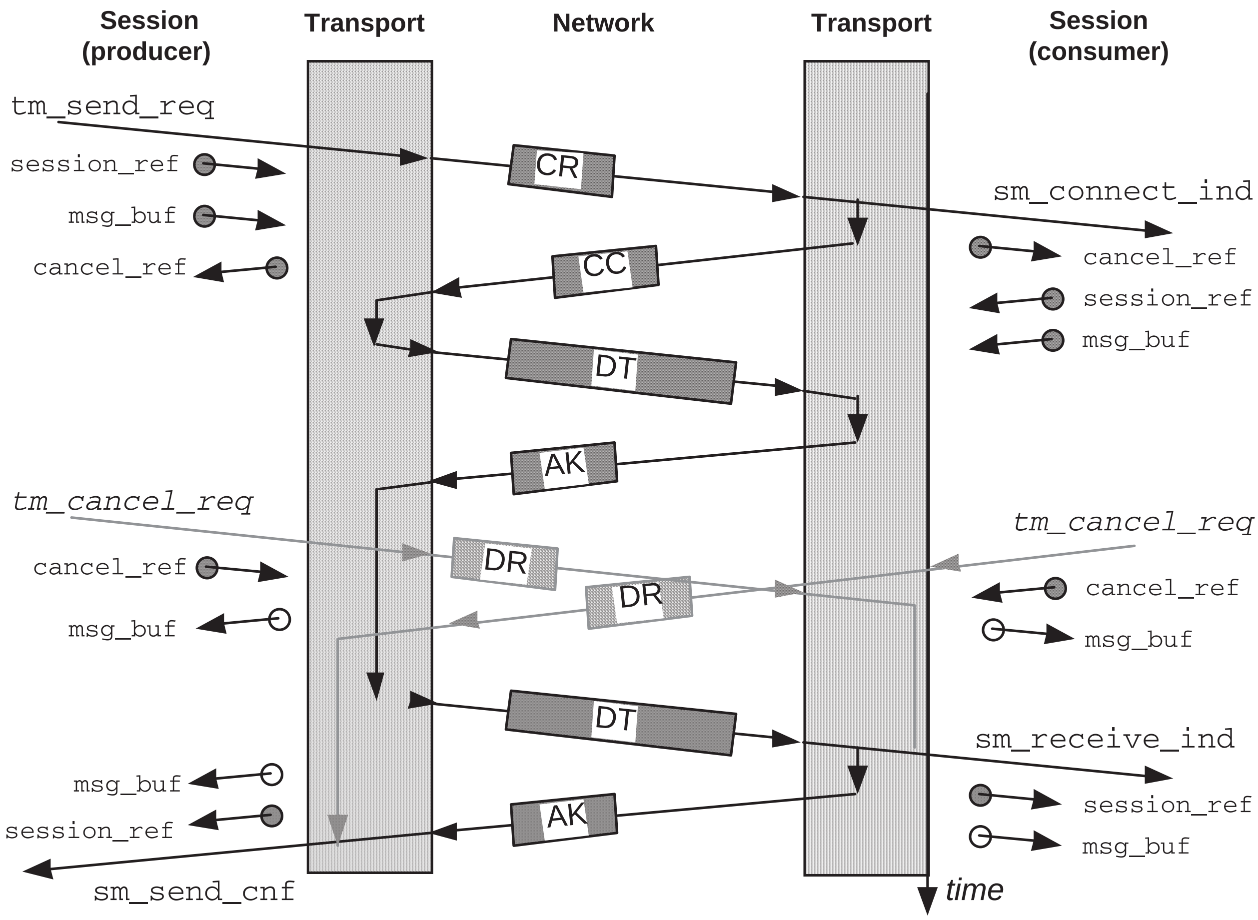
L’Interface de transport de messages (Transport_Message_Interface) fournit les services de la couche transport à la Couche Session.

Cette interface peut rester à l'intérieur d'un dispositif. Cette interface ne fait pas l'objet d’essais de conformité.

Les spécifications suivantes ont été incluses pour améliorer la portabilité. Les paragraphes qui suivent n'impliquent pas une mise en œuvre particulière. Toute interface qui fournit la même sémantique est autorisée.

6.3.6.7.1 Echange des données au niveau du transport

La Figure 132 montre l'interaction entre la couche transport et la Couche Session. Les cercles avec flèches indiquent le sens dans lequel les paramètres sont transmis.



Légende

Anglais	Français
session (producer)	session (producteur)
transport	transport
network	réseau
session (consumer)	session (consommateur)
time	durée

Figure 132 – Interface de transport

Le Producteur envoie un message contenu dans un msg_buf en appelant tm_send_req, identifiant la connexion par une référence de session session_ref.

A l'issue du transfert (réussi ou non), la couche transport appelle sm_send_cnf (avec la même session_ref) qui libère alors le tampon de message.

Chez le Consommateur, la couche transport appelle sm_connect_ind pour informer la Couche Session de la Connect_Request. Si la Couche Session accepte la Connect_Request, elle fournit un tampon msg_buf à la couche transport pour placer le message.

La couche transport appelle sm_received_ind lorsque la totalité du message a été reçu (ou annulé par l'autre partie). Cet appel contient session_ref.

Chaque partie peut annuler le transfert de message en appelant tm_cancel_req.

Les primitives de la TMI sont précédées du préfixe tm_.

La couche transport appelle les procédures de la Couche Session (avec le préfixe sm_) qui étaient déjà souscrites et dont le type est défini par la couche transport (préfixe TM_).

La TMI est définie par les constantes, les types et les procédures figurant dans le Tableau 47 et spécifiés dans les paragraphes suivants.

Tableau 47 – Primitives TMI

Nom	Signification
	Constantes
TM_CALLER_USER	l'utilisateur est un Appelant
TM_REPLIER_USER	l'utilisateur est un Répondeur
	Types
TM_MSG_DESCR	descripteur de message
TM_CONV_ID	indicatif de conversation
	Procédure d'initialisation
tm_define_user	annonce un utilisateur
	Procédures du Producteur
tm_send_req	envoie un message
TM_SEND_CNF	confirme envoi, procédure d'indication, du type de sm_send_cnf
sm_send_cnf	appelé quand le message est arrivé (ou a été annulé)
	Procédures du Consommateur
TM_CONNECT_IND	type de sm_connect_ind
sm_connect_ind	indique l'arrivée d'une Connect_Request
TM_RECEIVE_IND	type de sm_receive_ind
sm_receive_ind	appelé quand la totalité d'un message a été reçue
	Procédures du Producteur et du Consommateur
tm_cancel_req	annule le message

6.3.6.7.2 Type TM_MSG_DESCR

Définition	Type d'un message
Syntaxe	`typedef struct {} TM_MSG_DESCR;`
Usage	structure cachée utilisée comme indicatif pour nettoyer les structures de données verrouillées.

6.3.6.7.3 Type TM_CONV_ID

Définition	Type d'un Conversation_Id

Syntaxe	`typedef struct str_conv_id`
	`{`
	`UNSIGNED8 my_fct_or_station;`
	`UNSIGNED8 node;`
	`UNSIGNED8 function_or_station;`
	`UNSIGNED8 next_station`
	`} TM_CONV_ID;`

6.3.6.7.4 Procédure tm_define_user

Définition	Souscrit les procédures d'indication d'un utilisateur identifié comme Appelant ou Répondeur	

Syntaxe	`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`
	`);`	

Entrée	Who	utilisateur de la couche transport, (TM_CALLER_USER ou TM_REPLIER_USER)
	sm_connect_ind	procédure du Consommateur à appeler à l'arrivée d'un paquet Connect_Request. Voir définition du type TM_CONNECT_IND.
	sm_receive_ind	procédure du Consommateur à appeler lorsque la totalité d'un message a été reçue, annulée par le Producteur ou en cas d'erreur. Voir définition du type TM_RECEIVE_IND.
	sm_send_cnf	procédure du Producteur à appeler lorsque la totalité d'un message a été reçue, annulée par le Consommateur ou en cas d'erreur. Voir définition du type TM_SEND_CNF.
Renvoi		toute valeur de AM_RESULT

6.3.6.7.5 Procédure tm_send_req

Définition	Demande le transfert d'un message	

Syntaxe	`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`
	`);`	

Entrée	session_user	TM_CALLER_USER (Call_Message) ou TM_REPLIER_USER (Reply_Message)
	Conversation	Conversation_Id (concaténation de l'adresse de l'Appelant et du Répondeur)
	msg_addr	pointeur vers le corps du message
	msg_len	taille totale du corps du message
	hdr_addr	pointeur vers Session_Header
	hdr_len	taille totale de Session_Header
	session_ref	référence de session reliant tm_send_req au sm_send_cnf correspondant.
	my_topo	valeur du Topo_Counter fournie par l'application ou 0 si inconnue (= passe-partout)
Renvoi		toute valeur de AM_RESULT
Sortie	cancel_ref	indicatif des structures de données verrouillées de la couche transport.
Usage	1 – Le tampon du message à envoyer et le tampon pour placer son Session_Header sont séparés. 2 – session_ref est fournie par l'utilisateur de la couche transport et renvoyée par cette dernière lors de l'appel de sm_send_cnf. 3 – cancel_ref permet de libérer les structures de données verrouillées. cancel_ref n'est plus valide après une annulation réussie ou après le retour de sm_send_cnf.	

6.3.6.7.6 Type TM_SEND_CNF

Définition	Lorsque la totalité d'un message a été reçue ou annulée par le Consommateur ou pour signaler une erreur, la couche transport doit appeler la procédure de la Couche Session sm_send_cnf, qui est de ce type.	

Syntaxe	`typedef void ( * TM_SEND_CNF )`
	`(`
	`void * /* session_ref */`
	`UNSIGNED8 /* my_topo */`
	`AM_RESULT status`
	`);`	

Entrée	session_ref	référence de session qui associe sm_send_cnf avec le tm_send_req précédent
	my_topo	si le résultat est AM_OK, Topo_Counter est valide sur ce nœud
	Status	AM_OK si le Consommateur a accusé réception du message, sinon un code erreur est généré.
Usage	1 – Pour chaque tm_send_req, la couche transport appelle la procédure Couche Session sm_send_cnf, qui est du type TM_SEND_CNF, pour signaler au Producteur que son message a été reçu ou annulé par le Consommateur, ou pour signaler une erreur. 2 – sm_send_cnf n'est pas appelé si la connexion a bien été annulée par le Producteur. 3 – Status est un paramètre d'entrée.	

6.3.6.7.7 Type TM_CONNECT_IND

<table>
<tr><td rowspan="1">Définition</td><td colspan="2">Lorsqu'elle reçoit un paquet Connect_Request, la couche transport du côté Consommateur doit appeler la procédure sm_connect_ind de type TM_CONNECT_IND.
Si sm_connect_ind accepte le message, il envoie un résultat AM_OK et fournir un tampon pour le message.
Si sm_connect_ind renvoie un résultat autre que AM_OK, la couche transport doit rejeter la connexion et envoyer une Disconnect_Request en utilisant le résultat de sm_connect_ind comme paramètre ‘reason’.</td></tr>
<tr><td>Syntaxe</td><td colspan="2"><pre>
typedef void   (* TM_CONNECT_IND)
                   (
const   /* in: conversation   */
TM_CONV_ID*,   /* out: msg_addr   */
void * *,   /* in: msg_len */
UNSIGNED32,   /* out hdr_addr*/
void * *,   /* out: hdr_len*/
UNSIGNED *,   /* in: cancel_ref */
TM_MSG_DESCR *,   /* out: session_ref   */
void * *,   /* in: my_topo */
UNSIGNED8,   /* out: status */
AM_RESULT   );
</pre></td></tr>
<tr><td rowspan="4">Entrée</td><td>Conversation</td><td>Conversation_Id
(concaténation de l'adresse de l'Appelant et du Répondeur)</td></tr>
<tr><td>msg_len</td><td>taille du tampon de message fournie par la Couche session</td></tr>
<tr><td>cancel_ref</td><td>indicatif des structures de données verrouillées dans la couche transport</td></tr>
<tr><td>my_topo</td><td>Topo_Counter sur le nœud au moment où la Connect_Request a été reçue.</td></tr>
<tr><td rowspan="5">Sortie</td><td>msg_addr</td><td>pointeur vers le corps du message fourni par la Couche Session</td></tr>
<tr><td>hdr_addr</td><td>pointeur vers le Session_Header du message.</td></tr>
<tr><td>hdr_len</td><td>taille totale de Session_Header</td></tr>
<tr><td>session_ref</td><td>fourni par la Couche Session pour identifier les structures de données verrouillées dans la Couche Session. Associe sm_connect_ind au sm_receive_ind correspondant.</td></tr>
<tr><td>Status</td><td>AM_OK si la Couche Session accepte la connexion, sinon raison du rejet (AM_RESULT).</td></tr>
<tr><td>Usage</td><td colspan="2">La Couche Session souscrit une procédure sm_connect_ind, de type TM_CONNECT_IND, dans la procédure tm_define_user.</td></tr>
</table>

6.3.6.7.8 Type TM_RECEIVE_IND

Définition	Lorsque la couche transport a reçu un message entrant complet, elle doit appeler la procédure sm_receive_ind de type TM_RECEIVE_IND'.	

Syntaxe	`typedef void`
	
	`void *,`
	`AM_RESULT`
		`( * TM_RECEIVE_IND )`
		`(`
		`/* session_ref */`
		`/* status */`
		`);`

Entrée	session_ref	référence qui associe sm_receive_ind au sm_connect_ind précédent.
	Status	AM_OK lorsque le message a été bien reçu et placé dans le tampon, sinon un autre code d'erreur AM_RESULT.
Usage	1 – La Couche Session fournit une procédure sm_receive_ind de type TM_RECEIVE_IND, dans la procédure tm_define_user. 2 – sm_receive_ind retourne à la Couche session les tampons fournis par la Couche Session dans son sm_connect_ind.	

6.3.6.7.9 Procédure tm_cancel_req

Définition	Annule une connexion lorsqu'elle est appelée par le Producteur ou le Consommateur.	

Syntaxe	`AM_RESULT`
	
	`TM_MSG_DESCR *`
		`tm_cancel_req`
		`(`
		`cancel_ref`
		`);`

Entrée	cancel_ref	indicatif des structures de données verrouillées de la couche transport.
Renvoi		AM_OK si l'annulation concerne un message en cours, AM_FAILURE sinon.
Usage	1 – Si le résultat est AM_OK, la couche transport envoie un paquet Disconnect_Request et sm_send_cnf ou sm_receive_ind n'ont pas été appelés. 2 – Si la totalité du message a déjà été envoyé et si sa réception a été accusée, cette procédure n'a pas d'effet. 3 – cancel_ref permet de libérer les structures de données verrouillées, il n'est plus défini suite à une annulation réussie.	

6.3.7 Protocole de transport de distribution (en option)

6.3.7.1 Envoi de paquets de distribution

Le transport de distribution doit être divisé en trois phases:

a) l'établissement de la connexion;
b) le transfert de données avec accusé de réception;
c) la déconnexion.

La Figure 133 illustre un échange simple de paquets sans perte.

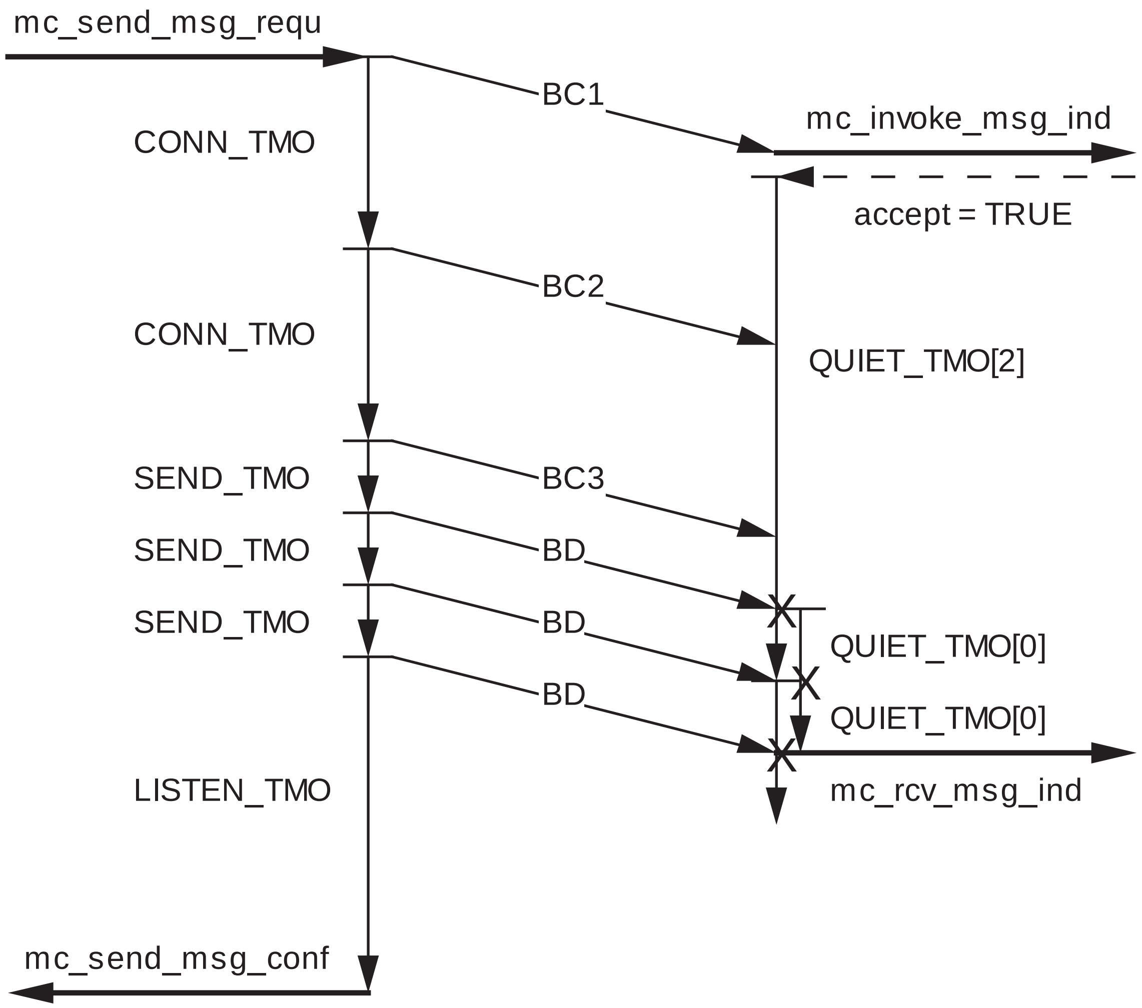


Figure 133 – Message distribué sans retransmission

6.3.7.1.1 Établissement de la connexion

Le premier paquet d'un message est un paquet Broadcast_Connect (BC) qui indique la taille totale du message et qui contient le premier segment de données.

A l'arrivée d'un paquet BC, le Consommateur vérifie s'il y a une application prête à recevoir le message. Si ce n'est pas le cas, le paquet est rejeté. Sinon, la connexion est ouverte et le premier segment de données est copié dans le tampon de message de l'application.

Le Producteur renvoie toujours le paquet BC trois fois avec une temporisation longue (CONN_TMO) entre les retransmissions.

Il suffit que le Consommateur reçoive l'une des trois copies.

Le paquet BC contient également un numéro de décompte de retransmission qui permet aux Consommateurs de régler leur temporisation.

Les Consommateurs ne peuvent pas demander la répétition de BC.

Si le message est assez court pour tenir à l'intérieur de BC, la situation représentée dans la Figure 134 s'applique:

- le Messager Producteur confirme l'achèvement du transfert immédiatement après avoir envoyé le dernier BC;

- Le Messager Consommateur en informe l'application et démarre une temporisation (ALIVE_TMO []) pour supprimer les doublons du message générés par la répétition des BC. Cette temporisation dépend du compte de répétition de BC. La réception du dernier BC arrête la temporisation et ferme la connexion.

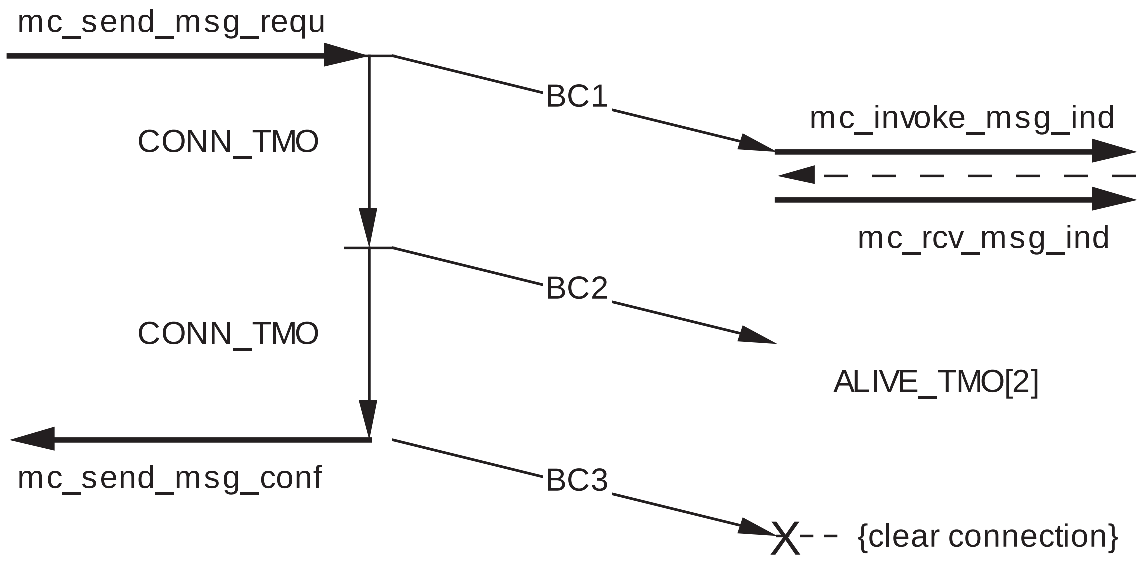


Figure 134 – Message distribué court sans paquet BD et sans perte

6.3.7.1.2 Transfert de données

Si le message n'a pas encore le BC, le Producteur commence à transmettre les paquets Broadcast_Data (BD) à une fréquence SEND_TMO qui est plus courte que CONN_TMO.

Les paquets BD contiennent un décalage de 16 bits qui indique la position du segment de données à l'intérieur du message entier.

Le Consommateur qui attend des données démarre une temporisation QUIET_TMO pour superviser l'activité. Si la temporisation expire ou si le Consommateur détecte un paquet dont le décalage est plus élevé que prévu, il considère que des paquets BD ont été perdus et envoie un paquet Broadcast_Repeat (BR) pour demander une retransmission. Les paquets BR contiennent un décalage de 16 bits à partir desquels la transmission est demandée.

Pendant la transmission des paquets BD, le Producteur filtre les paquets BR entrants et commence la retransmission après avoir introduit une pause de transmission (PAUSE_TMO en plus du SEND_TMO normal).

Pendant la transmission du BR, le Consommateur démarre un REPEAT_TMO pour superviser l'effet du BR.

Le Consommateur ne peut envoyer un BR que si le REPEAT_TMO n'est pas encore en cours, sinon REPEAT_TMO risque d'expirer, provoquant la retransmission de BR, tandis que le message serait rejeté lors d'une deuxième temporisation et la connexion fermée (dans ce cas, trois paquets auraient été perdus).

La réception du BD attendu ou plus ancien redémarre la temporisation QUIET_TMO [0].

EXEMPLE – La Figure 135 illustre une situation de perte de paquets.

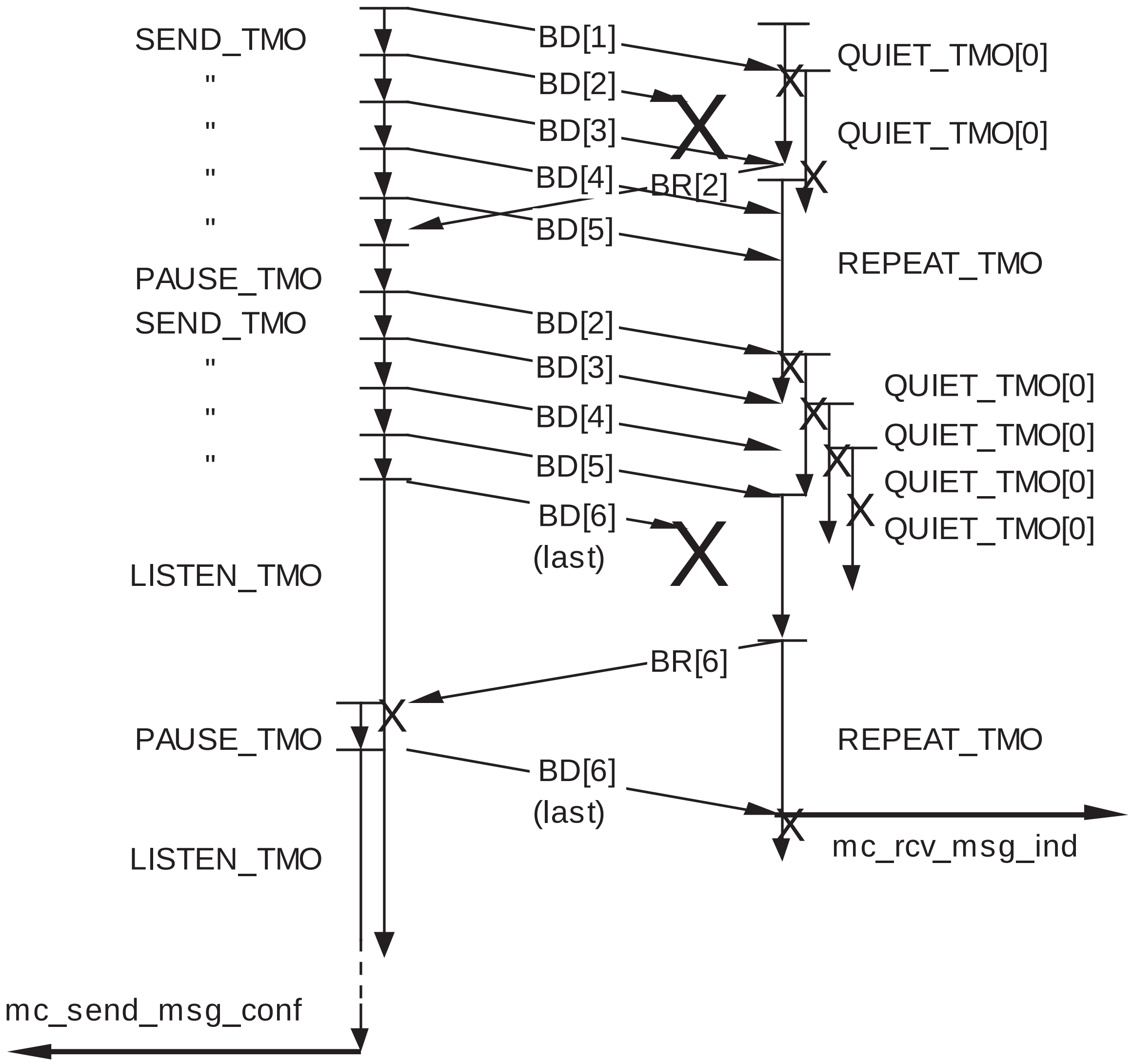


Figure 135 – Echange avec des paquets perdus

6.3.7.1.3 Déconnexion

Après le dernier BD, le message est transmis à l'application Consommateur et la connexion est fermée.

Après le dernier paquet BD, le Producteur attend les paquets Broadcast_Repeat (BR) pendant LISTEN_TMO avant de fermer et de confirmer la fin du transfert.

6.3.7.1.4 Annulation d'un transfert

Si le Consommateur annule une demande, un message entrant est rejeté et rien n'est visible sur le bus. En revanche, si un Producteur annule une distribution de message, il est préférable d'émettre un paquet particulier (Broadcast_Stop, BS) pour empêcher tous les Consommateurs de dépasser leur temporisation et de générer une grande quantité de paquets répétition BR.

6.3.7.1.5 Algorithme de retransmission

Le même paquet BR peut être émis par plusieurs Consommateurs, mais ces paquets arrivent chez le Producteur en différé. Il se peut également que les paquets BR entrants n'indiquent pas un décalage manquant croissant (dans le cas du transfert point à point). Ce problème ne peut pas être résolu en insérant un retard long avant de commencer la retransmission, puisque les temporisations de chacun des deux côtés dépendent de l'autre.

Le Producteur a donc besoin d'un certain mécanisme de filtre BR. Le filtre utilisé peut être défini de la manière suivante:

- une fois qu'un paquet BR a été accepté, un autre paquet BR entrant avec le même décalage de retransmission est rejeté pendant une temporisation SKIP_TMO;
- seul un nombre limité REP_LIMIT de paquets BR à décalages de retransmission différents est assuré d'être accepté à l'intérieur d'une fenêtre glissante de décalage d'une taille fixe REP_RANGE;
- des paquets BR supplémentaires qui dépassent cette limite pour toute fenêtre peuvent être rejetés ou non.

Les paquets BR entrants sont traités selon l'algorithme suivant: le Producteur tient un tableau d'enregistrement des retransmissions demandées pour chaque message. Ce tableau comprend les entrées REP_LIMIT qui sont toutes vides au début du transfert. Une entrée occupée contient le décalage de retransmission (une clé) et son propre temporisateur chargé avec SKIP_TMO lorsqu'un BR correspondant est accepté. A l'arrivée d'un BR, le tableau d'enregistrement est contrôlé pour vérifier si le décalage de retransmission demandé a déjà été enregistré. Si une telle entrée existe avec le même décalage, le BR est rejeté si le temporisateur de cette entrée est en cours d'exécution, sinon il est accepté. Si le décalage demandé n'est pas encore enregistré, le BR est alors accepté et une nouvelle entrée est insérée tant que le tableau n'est pas saturé. Une fois le tableau saturé, l'acceptation d'un BR dépend de l'existence d'une entrée qui peut être libérée pour être réutilisée. L'entrée au décalage le plus bas est écrasée si le décalage du nouveau BR ou le décalage d'une autre entrée n'entre pas dans une fenêtre dont la limite inférieure correspond au décalage enregistré le plus bas. Sinon le BR est rejeté car les demandes BR REP_LIMIT à l'intérieur de cette fenêtre ont déjà été acceptées. Une fois le tableau d'enregistrement saturé, il le reste définitivement (jusqu'à la fermeture de la connexion).

Grâce à cet algorithme et avec un seul Consommateur du message, tous les paquets BR qui dépassent la limite REP_LIMIT sont rejetés, quelle que soit la fenêtre (s'il y a plusieurs Consommateurs, un tel BR peut toujours être accepté).

6.3.7.1.6 Cohérence des messages à travers une inauguration

Le Consommateur d'un message MC permet de sauvegarder le Final_Node qui se trouve dans l'en-tête réseau d'un paquet BC entrant. Une fois qu'une connexion de réception est ouverte, les paquets suivants ne peuvent être acceptés que s'ils sont adressés au même Final_Node, sinon la connexion est fermée et une erreur est générée. Le nœud de routage vérifie que le champ Final_Node des paquets envoyés au bus de véhicule change à chaque inauguration (le champ Final_Node contient un compteur d'inauguration). Cela empêche le regroupement des paquets provenant de nœuds différents en un message mixte invalide (l'adresse d'un nœud peut être attribuée à un autre nœud par une inauguration, et les deux nœuds peuvent accidentellement utiliser la même Connection_Reference).

Si le Producteur d'un message MC se trouve sur un dispositif final du bus de véhicule, il n'est pas au courant d'une inauguration intermédiaire et continue à envoyer les paquets BD tant que la totalité du message n'a pas été transmise.

6.3.7.2 Identification de la connexion

Pour chaque paquet entrant, la connexion à laquelle le paquet appartient est récupérée. Une connexion est identifiée chez le récepteur par une structure à 32 bits comprenant sa propre Final_Function, l'Origin_Function, l'Origin_Node ainsi que l'Origin_Station.

Chez l'émetteur, une connexion est identifiée uniquement par les deux fonctions concernées, puisque les paquets BR entrants peuvent comprendre n'importe quel nœud d'origine (il y a plusieurs Consommateurs).

Les messages ayant la même identification de connexion sont transmis en séquence.

6.3.7.3 Référence de connexion

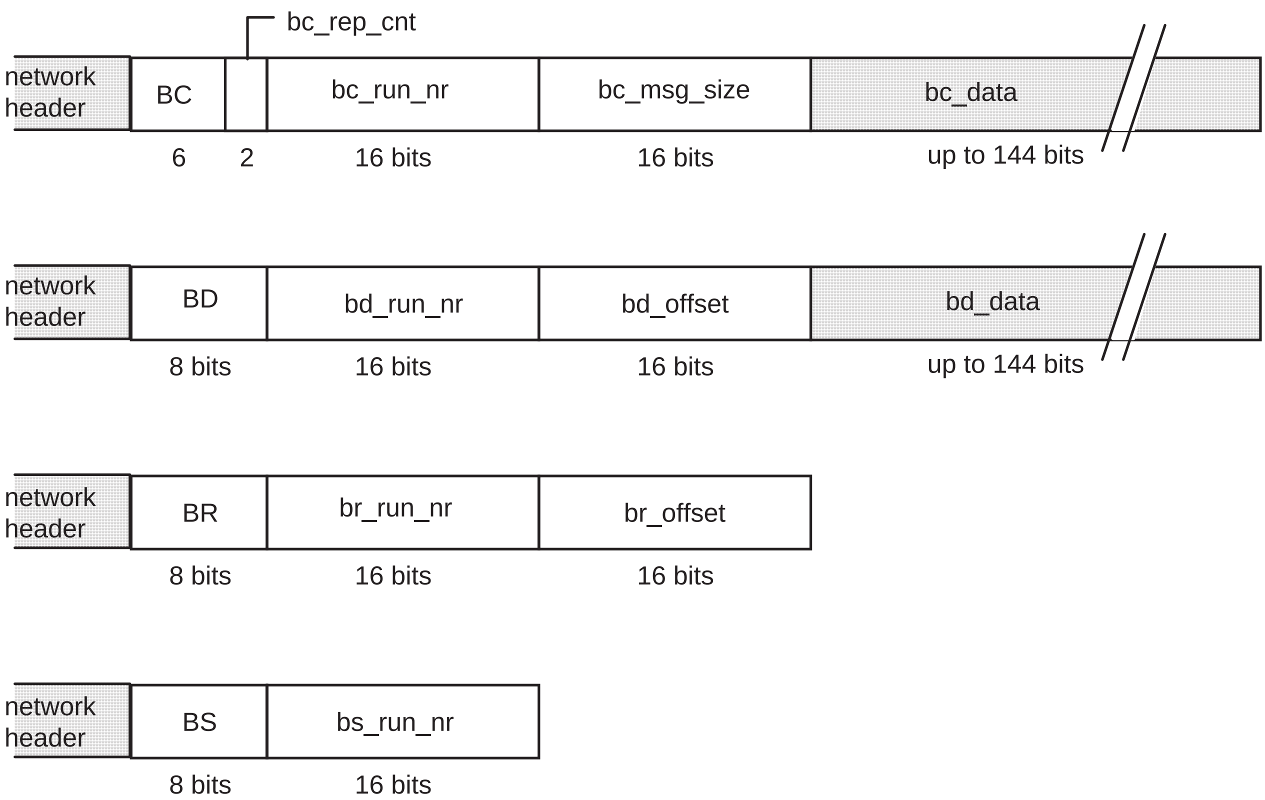
Chaque paquet contient une Connection_Reference à 16 bits pour identifier le message. Cette identification est attribuée par le Producteur à chaque message sortant en ordre séquentiel et initialisée de manière aléatoire.

6.3.7.4 Codage du contrôle de transport de messages distribués

Le Protocole MCP utilise la même couche réseau que le protocole MTP. Les paquets ont donc les mêmes Link_Header et Network_Header. Le Contrôle de Transport des Messages, qui est spécifié en 6.3.6.5.2, fait la distinction entre les deux protocoles.

6.3.7.5 Codage des paquets distribués

Le Protocole MCP fait la distinction entre les paquets suivants (voir la Figure 136).



Légende

Anglais	Français
network header	en-tête de réseau
up to	jusqu'à

Figure 136 – Format de paquet

Les paquets distribués doivent être formatés comme indiqué en 6.3.6.5.4.

6.3.7.6 Définition du protocole de transport MCP

Ce protocole MCP fonctionne sur un réseau sans connexion non fiable qui maintient la séquence de paquets et limite la durée de vie des paquets à PACK_LIFE_TIME.

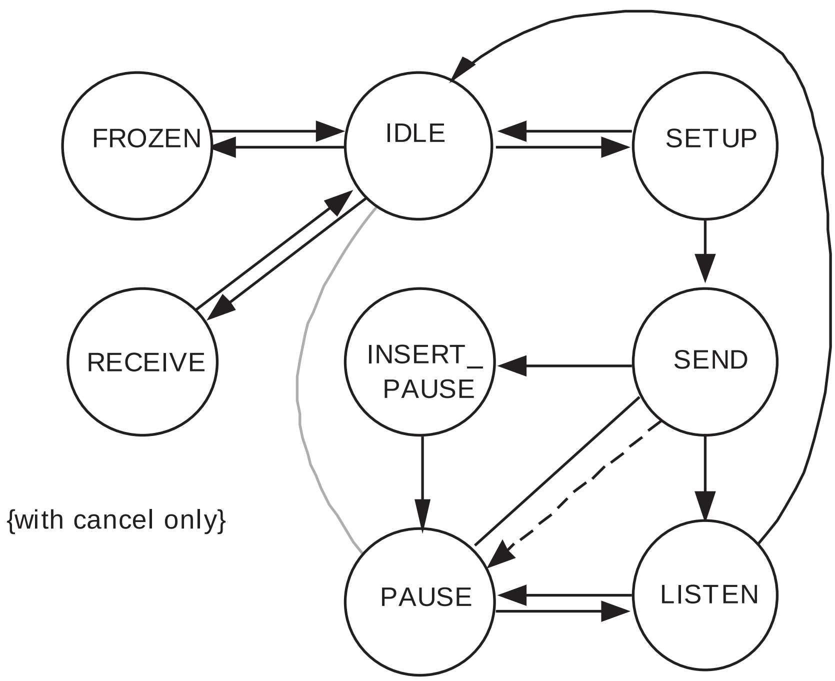
Une connexion est établie pour chaque transfert de message. Le Producteur du message s'appelle Producteur, tandis que le partenaire récepteur s'appelle Consommateur. Un temporisateur est nécessaire pour chaque connexion chez le Consommateur, tandis que le Producteur a besoin d'un temporisateur supplémentaire pour chaque entrée du tableau d'enregistrement pour la mise en œuvre de SKIP_TMO.

6.3.7.6.1 Etats et diagramme de transition d'état

La machine de protocole MCP, illustrée à la Figure 137, doit avoir les états donnés dans le Tableau 48.

Tableau 48 – Etats de la machine MCP

Nom d'état	Concerne	Description brève de l'état
IDLE	Producteur, Consommateur	déconnecté
SETUP	Producteur	envoi de paquets BC
SEND	Producteur	envoi périodique de paquets BD
LISTEN	Producteur	message complet envoyé, en attente de paquets BR
INSERT_PAUSE	Producteur	délai supplémentaire demandé avant d'envoyer le paquet BD suivant
PAUSE	Producteur	délai supplémentaire en cours
RECEIVE	Consommateur	paquet BC reçu et paquets BD attendus
FROZEN	Consommateur	certains paquets BC reçus, mais pas le dernier, et aucun paquet BD n'est attendu



Légende

Anglais	Français
(with cancel only)	(avec annulation uniquement)

Figure 137 – Etats de la machine de protocole

Les transitions entre ces états dépendent des événements entrants définis dans le Tableau 49 et génèrent les événements sortants définis dans le Tableau 50.

Tableau 49 – Evénements entrants

Événement	Concerne	Interface	Description brève des événements
mc_send_msg_requ	Producteur	de l'utilisateur	demande d'envoi d'un message, l'utilisateur envoie un tampon de message au Messager
mc_cancel_msg	Producteur, Consommateur	de l'utilisateur	annule un transfert de message entrant ou sortant le Messager renvoie le tampon de message à l'utilisateur si l'annulation aboutit
rcv_BC	Consommateur	du réseau	paquet BC reçu
rcv_BD	Consommateur	du réseau	paquet BD reçu
rcv_BS	Consommateur	du réseau	paquet BS reçu
rcv_BR	Producteur	du réseau	paquet BR reçu
skipped (reg_entry)	Producteur	interne	SKIP_TMO a expiré pour l'entrée reg_entry du tableau d'enregistrement
TMO	Producteur, Consommateur	interne	toute autre temporisation a expiré

Tableau 50 – Evénements sortants

Événement	Concerne	Interface	Description brève des événements
mc_invoke_msg_ind	Consommateur	à l'utilisateur	l'utilisateur doit accepter ou rejeter la Connect_Request entrante, l'utilisateur envoie un tampon de message au Messager si l'utilisateur accepte le message
mc_rcv_msg_ind	Consommateur	à l'utilisateur	message complet reçu ou erreur reçue le Messager renvoie le tampon de message à l'utilisateur
mc_send_msg_conf	Producteur	à l'utilisateur	message complet envoyé ou erreur envoyée le Messager renvoie le tampon de message à l'utilisateur
send_BC	Producteur	au réseau	transmission d'un paquet BC
send_BD	Producteur	au réseau	transmission d'un paquet BD
send_BS	Producteur	au réseau	transmission d'un paquet BS
send_BR	Consommateur	au réseau	transmission d'un paquet BR

6.3.7.6.2 Champs de contrôle dans les paquets de données

Les paquets échangés contiennent les champs de contrôle décrits dans le Tableau 51.

Tableau 51 – Champs de contrôle des paquets

Événement	Noms des champs	Description brève des champs
xxx_BC (	BC_run_nr,	Connection_Reference
	BC_rep_cnt,	compte des retransmissions BC en attente
	BC_msg_size)	taille du message complet
xxx_BD (	BD_run_nr,	Connection_Reference
	BD_offset)	décalage du segment de données à l'intérieur du message
xxx_BS (	BS_run_nr)	Connection_Reference
xxx_BR (	BR_run_nr, BR_offset)	Connection_Reference décalage manquant

NOTE L'abréviation xxx signifie 'send' ou 'rcv'. Si xxx = send events, les paramètres réels sont spécifiés pour les champs de contrôle, tandis que si xxx = rcv events, les actions donnent la référence des noms formels des champs.

6.3.7.6.3 Variables auxiliaires

Le diagramme de transition d'état dépend des variables auxiliaires pour réduire le nombre d'états. Certaines variables existent uniquement chez le Producteur ou le Consommateur et appartiennent à la connexion, tandis que d'autres sont générales, ce qui signifie qu'elles sont utilisées en commun pour toutes les connexions d'un site. Les variables auxiliaires données dans le Tableau 52 doivent être mises en œuvre.

Tableau 52 – Variables auxiliaires

Nom de la variable	Contexte	Description brève de la variable
run_nr	global	Connection_Reference libre suivante
cancelled	Producteur, Consommateur	ce message a été annulé par l'utilisateur
timer	Producteur, Consommateur	temporisateur pour toutes les temporisations, à l'exception de SKIP_TMO
msg_size	Producteur, Consommateur	taille du message complet
offset	Producteur, Consommateur	décalage du segment de données suivant
conn_run_nr	Producteur, Consommateur	Connection_Reference de cette connexion
rep_cnt	Producteur, Consommateur	nombre de répétitions (BR ou BC)
my_node	Consommateur	Final_Node pour cette connexion
reg_table[REP_LIMIT]. offset	Producteur	tableau d'enregistrement, décalages de retransmission
reg_table[REP_LIMIT]. skip_timer	Producteur	tableau d'enregistrement, temporisations SKIP_TMO
reg_table[REP_LIMIT]. timer_running	Producteur	tableau d'enregistrement, booléen: 'skip_timer' est en cours.
table_is_full	Producteur	booléen: le tableau d'enregistrement est plein.
window_is_full	Producteur	booléen: tous les décalages enregistrés se trouvent dans la même fenêtre dont la taille est REP_RANGE.
next_entry	Producteur	entrée (indice) libre suivante dans le tableau d'enregistrement ou entrée avec le décalage le plus faible si 'table_is_full'
NOTE Un temporisateur peut uniquement être démarré ou arrêté (mais non replanifié) et génère un événement quand il expire.		

6.3.7.6.4 Constantes

Le Tableau 53 détaille les constantes utilisées à l'intérieur de la machine de protocole MCP.

Tableau 53 – Constantes MCP

Nom de la constante	Valeur de la constante	Description brève de la constante
REP_LIMIT	6	taille du tableau d'enregistrement (nombre d'entrées)
MAX_TRIALS	3	nombre d'essais (paquets BC transmis), la perte d'un paquet est admise
BD_DATA_SIZE	18 (pour trame MVB)	nombre d'octets de données dans un paquet BD
PROC_DELAY	1 × 64 [ms]	temps de traitement des événements (période d'interrogation)
NET_DELAY	8 × 64 [ms]	délai de transmission prévu (bout en bout)
REP_RANGE	LISTEN_TMO / SEND_TMO × BD_DATA_SIZE	
NOTE La plage des écarts à l'intérieur de laquelle la retransmission est possible définit la taille de fenêtre REP_RANGE.		

Les temporisations doivent être celles spécifiées dans le Tableau 54.

Tableau 54 – Temporisations MCP

Nom de la temporisation	Valeur de temporisation
	Côté Producteur
CONN_TMO	4 × 64 [ms]
SEND_TMO	1 × 64 [ms]
PAUSE_TMO	1 × 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) × (REPEAT_TMO + PROC_DELAY) + 2 × NET_DELAY + PROC_DELAY – SEND_TMO
	Côté Consommateur
QUIET_TMO [MAX_TRIALS]	SEND_MAX_TMO + PROC_DELAY + NET_DELAY + ix × (CONN_TMO + PROC_DELAY)
ALIVE_TMO [MAX_TRIALS]	NET_DELAY + PACK_LIFE_TIME + ix × (CONN_TMO + PROC_DELAY)
REPEAT_TMO	SEND_MAX_TMO + 2 × NET_DELAY + PROC_DELAY
NOTE 1 Toutes les temporisations et les autres indications de temps sont spécifiées en millisecondes. Pour les temporisations qui peuvent être choisies librement, un numéro sélectionné est spécifié. Pour les temporisations dépendantes, une formule pour calculer la valeur minimale est donnée. NOTE 2 'ix' signifie 'array index 0 .. (MAX_TRIALS – 1)'.	

6.3.7.6.5 Actions composées

Les actions composées figurant dans le Tableau 55 sont utilisées plusieurs fois par la machine MCP.

Chaque connexion possède son propre temporisateur pour la supervision des temporisations. Le temporisateur est commandé par les actions restart_tmo et reset_tmo, et génère un événement interne TMO à l'expiration. Chez le Producteur, un temporisateur supplémentaire est utilisé pour chaque SKIP_TMO manipulé par restart_skip ou reset_skip. Il génère l'événement interne skipped (reg_entry) à l'expiration. Tous les temporisateurs appartenant à une connexion sont remis à zéro lors de la déconnexion.

Tableau 55 – Actions composées MCP

Nom de l'action	Description brève de l'action	État suivant
restart_tmo (xxx_TMO)	arrête le temporisateur et le démarre avec la valeur xxx_TMO;	
reset_tmo	arrête le temporisateur;	
restart_skip (reg_entry)	démarre le skip_timer avec SKIP_TMO et attribue la valeur TRUE à timer_running pour reg_table[reg_entry];	
reset_skip (reg_entry)	arrête skip_timer et attribue la valeur FALSE à timer_running pour 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;
		IDLE
		FROZEN
		RECEIVE

Le check_BR filtre les paquets BR et renvoie un booléen pour signaler si le BR a été accepté ou non. Le check_BR accède exclusivement à toutes les variables auxiliaires qui n'apparaissent que chez le Producteur, à l'exception de l'initialisation, réalisée par init_BR_filter. Le filtrage est spécifié dans le Tableau 56.

Tableau 56 – Filtrage des paquets BR

Nom de l'action	Spécification de l'action

init_BR_filter	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;

check_BR (VAR accept)	temporary variables: ix, max_offset;
	accept:= FALSE;
	IF ((BR_run_nr <> conn_run_nr) OR (BR_offset >= offset)) THEN
	RETURN;
	END;
	IF (any entry is 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;

6.3.7.6.6 Tableau des événements d'état du producteur

Le Tableau 57 spécifie le comportement du Producteur.

Tableau 57 – Tableau des événements d'état du Producteur MCP

État courant	Événement	Action(s)	État suivant
(quelconque)	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
			
			IDLE
			
			SEND

SEND	rcv_BR	check_BR (VAR accept);
		IF (accept) THEN
		
		END;
			
			INSERT_
			    PAUSE

	TMO AND cancelled	restart_tmo (PAUSE_TMO);	PAUSE

Tableau 57 *(suite)*

État courant	Événement	Action(s)	État suivant

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

6.3.7.6.7 Tableau des événements d'état du consommateur

Lors d'un événement, le Consommateur dans un état courant doit passer à l'état suivant et effectuer l'action spécifiée dans le Tableau 58.

Tableau 58 – Tableau des événements d'état du Consommateur MCP

État Actuel	Événement	Action(s)	État suivant
(quelconque)	mc_cancel_msg	cancelled:= TRUE;	
IDLE	rcv_BC	accept_BC;	IDLE OR FROZEN OR RECEIVE

RECEIVE	rcv_BD AND
	(BD_run_nr = conn_run_nr)
	AND
	(my_node = final_node)
		IF (cancelled) THEN
		reset_tmo;
		ELSIF (BD_offset = offset) THEN
		update (offset);
		IF (offset = msg_size) THEN
		close_rcv (AM_OK);
		ELSE
		rep_cnt:= 0;
		restart_tmo (QUIET_TMO[0]);
		END;
		ELSIF (BD_offset > offset) THEN
		IF (rep_cnt = 0) THEN
		send_BR (conn_run_nr, offset);
		INC (rep_cnt);
		restart_tmo (REPEAT_TMO);
		END;
		ELSE
		rep_cnt:= 0;
		restart_tmo (QUIET_TMO[0]);
		END;
			
			IDLE
			
			IDLE

	TMO AND
	(rep_cnt < (MAX_TRIALS – 1))
		IF (cancelled) THEN
		reset_tmo;
		ELSE
		send_BR (conn_run_nr, offset);
		INC (rep_cnt);
		restart_tmo (REPEAT_TMO);
		END;
			IDLE

	TMO AND (rep_cnt = (MAX_TRIALS – 1))	close_rcv (AM_REPEAT_TMO_ERR);	IDLE
	rcv_BC AND (BC_run_nr = conn_run_nr) AND (my_node = final_node)	(* no action *)	

Tableau 58 *(suite)*

État Actuel	Événement	Action(s)	État suivant
RECEIVE (suite)	(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

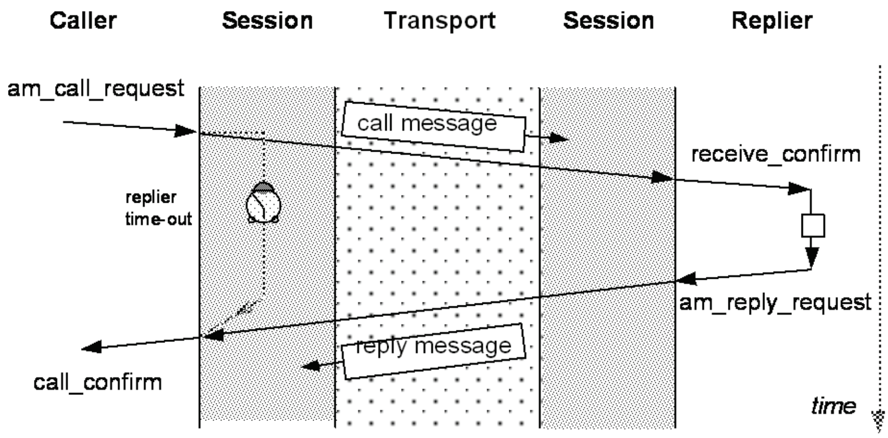
6.3.8 Couche session de messages

6.3.8.1 Objet

La Couche Session associe deux messages: un Call_Message et un Reply_Message.

Le Call_Message est envoyé par l'Appelant au Répondeur, le Reply_Message est envoyé par le Répondeur à l'Appelant.

Ce couple de messages permet la mise en œuvre d'un Appel de Procédure à distance (voir la Figure 138).



Légende

Anglais	Français
caller	appelant
session	session
transport	transport
replier	répondeur
call message	message d'appel
replier time-out	temporisation du répondeur
time	durée
reply message	message de réponse

Figure 138 – Transfert Couche Session

La Couche Session utilise les services de la couche transport pour chaque transfert de message.

6.3.8.2 Indicatif de conversation

La Couche Session doit identifier de manière unique les parties qui dialoguent par un Conversation_Id, qui est la concaténation de:

- la Network_Address de l'application distante;
- le Function_Id de l'application locale.

La Couche Session doit rejeter une demande de communication qui utilise un Conversation_Id donné, tant qu'une autre session avec le même Conversation_Id est en cours.

La Couche Session doit conserver l'adresse complète de l'Appelant pour envoyer le Reply_Message correspondant.

NOTE Le Conversation_Id contient toutes les informations nécessaires pour envoyer le paquet à la couche réseau et pour identifier les paquets en provenance de la couche réseau.

6.3.8.3 Raccourci

La Couche Session doit contourner le réseau lorsque le partenaire se trouve sur la même station, c'est-à-dire envoyer directement le message sans impliquer la couche transport.

6.3.8.4 Vérification du Topo_Counter

La Couche Session doit vérifier la cohérence entre my_topo (fourni par l'Application) et this_topo (la valeur du Topo_Counter conservée par la couche réseau) conformément à l'algorithme suivant:

```
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.
```

La Couche Session doit utiliser la valeur du Topo_Counter pendant toute la durée de la conversation.

NOTE Cela garantit l'annulation d'une conversation si une inauguration a lieu entre l'Appel et le Reply_Message.

6.3.8.5 Codage de l'en-tête session

Dans la Connect_Request d'un Call_Message, le Session_Header consiste en un octet rempli de '0'. Toutes les autres combinaisons sont réservées (voir la Figure 139).

Dans la Connect_Request d'un Reply_Message, le Session_Header consiste en un octet qui contient l'état de réponse fourni par l'application Répondeur.

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

Figure 139 – Session_Header de Call_Message (de type Am_Result)

6.3.8.6 Gestion des tampons

La Couche Session doit assurer la gestion de deux tampons:

a) les tampons statiques, attribués ou libérés par l'application, et

b) les tampons dynamiques, attribués ou libérés par la Couche Session.

6.3.8.7 Interface de la Couche Session

L'interface de la Couche Session est identique à celle de la Couche Application puisque la couche de présentation n'a pas de protocole.

6.3.9 Couche de présentation des messages

La couche de présentation des messages n'a pas de protocole.

Les messages doivent être transmis sous la forme ARRAY OF WORD8, en ordre croissant d'adresses mémoire.

Les en-têtes et paramètres de message doivent suivre les mêmes règles de présentation de données que pour les Process_Variables (les données sont transmises en commençant par l'octet de poids fort, par exemple).

Le paragraphe 6.4 liste les types de données qui doivent être utilisés dans les protocoles et qui sont préconisés pour l'application.

6.3.10 Couche d'application des messages

6.3.10.1 Objet

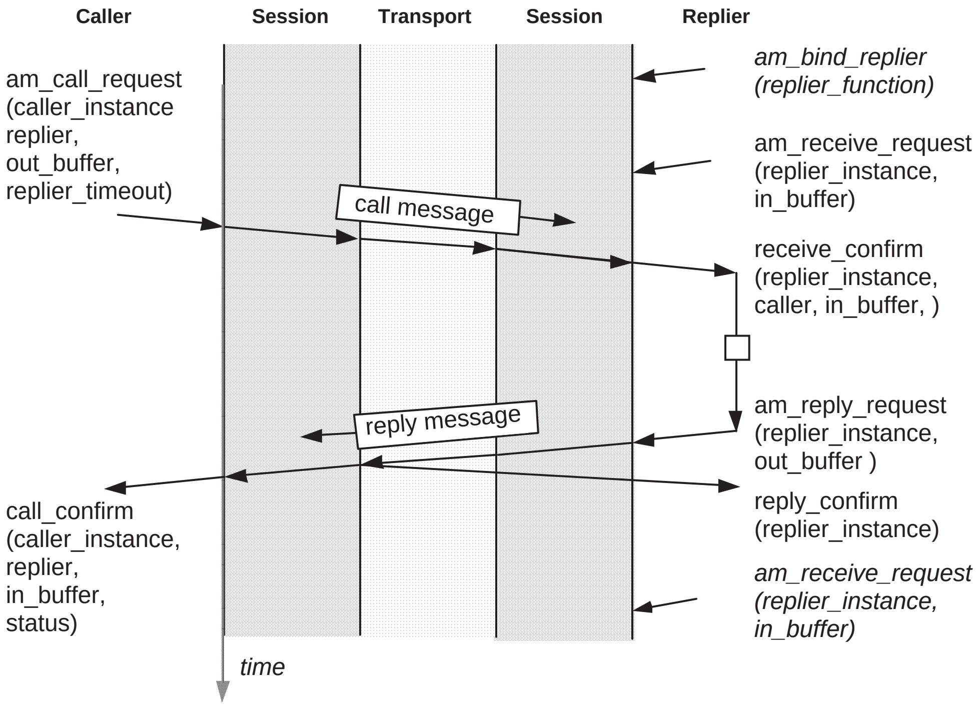
L'interface d'application des messages (AMI) permet à une application d'envoyer et de recevoir des messages sur le réseau. L'AMI propose un service d'Appel et de Réponse, l'initialisation et la gestion des tampons, ainsi qu'un service de distribution.

L'AMI est définie comme un ensemble de procédures qui accèdent directement à la Couche Session (la couche de présentation et la Couche Application n'ont pas de protocole).

6.3.10.2 Interface d'application de messages (AMI)

6.3.10.2.1 Primitives AMI

L'interface de la Couche Application doit mettre en œuvre les primitives illustrées à la Figure 140, détaillées dans le Tableau 59 et spécifiées dans les paragraphes suivants.



Légende

Anglais	Français
Caller	Appelant
Replier	Répondeur
call message	message d'appel
time	durée
reply message	message de réponse

Figure 140 – Interface d'application de messages

NOTE 1 Les objets AMI sont précédés du préfixe am_ ou AM_ (pour les messages d'application), les objets appartenant à l'instance de l'Appelant ou du Répondeur n'ont pas de préfixe.

NOTE 2 Les abréviations suivantes sont utilisées dans les noms:

REM - défaillance à distance signalée par le dispositif partenaire;

LOC - défaillance locale signalée par le dispositif lui-même;

OVF - débordement;

TMO - temporisation.

Tableau 59 – Primitives AMI

Nom	Signification
	Constantes et Types
AM_RESULT	résultat d'une procédure, même définition que Am_Result.
AM_ADDRESS	Adresse Réseau de l’entité éloignée
	Initialisation
am_init	initialise le Messager
am_announce_device	configure un dispositif
am_show_busses	donne le Bus_Id des couches de liaisons connectées
am_set_current_tc	indique au Messager le Topo_Counter courant
	Interface du répertoire de stations
AM_STADI_ENTRY	entrée du répertoire de stations
am_stadi_write	écrit le répertoire de stations
am_stadi_read	lit le répertoire de stations
	Interface du répertoire de fonctions
AM_DIR_ENTRY	élément du répertoire de fonctions
am_clear_dir	initialise le répertoire de fonctions
am_insert_dir_entries	enregistre l’indicatif de station d’une liste de fonctions
am_remove_dir_entries	efface une liste de fonctions
am_get_dir_entry	récupère l’indicatif de station d’une fonction donnée
	Interface du répertoire de groupes
AM_GROUP	définition d’un groupe
am_clear_groups	efface le répertoire de groupes
am_insert_member	insère un nœud dans un répertoire de groupes
am_remove_member	efface un nœud d’un répertoire de groupes
am_member	membre d’un répertoire de groupes
	Interface Appelant
am_call_request	l’Appelant envoie un message complet
AM_CALL_CONFIRM	type de la procédure appelée à l’arrivée de la réponse
am_call_cancel	annule la conversation et rejette le Reply_Message
	Interface Répondeur
am_bind_replier	annonce une instance Répondeur à la Couche Session
am_unbind_replier	annule l’annonce ci-dessus
am_receive_request	l’instance de l’annonce est prête pour l'appel suivant
AM_RECEIVE_CONFIRM	type de la procédure appelée lorsque l’appel est achevé
am_reply_request	appelé par l’instance Répondeur pour renvoyer un Reply_Message
AM_REPLY_CONFIRM	type de la procédure appelée lorsque la réponse est envoyée
am_receive_cancel	annule une instance répondeuse prête ou occupée
	Traitement des tampons
am_buffer_free	recycle un tampon dynamique de messages

NOTE Les procédures d’interface ne sont pas bloquantes et l'ordonnancement des tâches n’est pas contraint.

6.3.10.2.2 Définition d'AM_RESULT

Définition	Une procédure qui renvoie une valeur de type AM_RESULT doit la coder de la manière suivante:
Syntaxe	(voir ci-dessous)

```
typedef enum
  {
  AM_OK                  =0, /* accomplissement correct
  AM_FAILURE             =1, /* défaillance non spécifiée
  AM_BUS_ERR             =2, /* transmission bus non possible
  AM_REM_CONN_OVF        =3, /* trop de connections entrantes
  AM_CONN_TMO_ERR        =4, /* Connect_Request non répondue
  AM_SEND_TMO_ERR        =5, /* Envoyer temporisation (Connect
réussi)
  AM_REPLY_TMO_ERR       =6, /* réponse non reçue
  AM_ALIVE_TMO_ERR       =7, /* message pas reçu en entier
  AM_NO_LOC_MEM_ERR          =8,  /* manque de mémoire ou de
temporisateurs
  AM_NO_REM_MEM_ERR          =9,  /* partenaire manque de mémoire
ou de temporisateur
  AM_REM_CANC_ERR        =10,/* annulé par le partenaire
  AM_ALREADY_USED        =11,/* même opération déjà effectuée
  AM_ADDR_FMT_ERR        =12,/* erreur de format d'adresse
  AM_NO_REPLY_EXP_ERR    =13,/* réponse innattendue
  AM_NR_OF_CALLS_OVF     =14,/* trop d'appels reçus
  AM_REPLY_LEN_OVF       =15,/* message de réponse trop long
  AM_DUPL_LINK_ERR       =16,/* erreur conversation dupliquée
  AM_MY_DEV_UNKNOWN_ERR      =17, /* mon adresse est inconnue ou
invalide
  AM_NO_READY_INST_ERR   =18,/* répondeur non prêt
  AM_NR_OF_INST_OVF          =19, /* trop d'instances répondeur
  AM_CALL_LEN_OVF        =20,/* message d'appel trop long
  AM_UNKNOWN_DEST_ERR    =21,/* dispositif du partenaire inconnu
  AM_INAUG_ERR           =22,/* inauguration du train a eu lieu
  AM_TRY_LATER_ERR       =23,/* (pour usage interne)
  AM_FIN_NOT_REG_ERR     =24,/* adresse finale non enregistrée
  AM_GW_FIN_NOT_REG_ERR      =25, /* adresse finale inconnue dans
le routeur
  AM_GW_ORI_REG_ERR          =26, /* adresse origine inconnue
dans le routeur
  AM_MAX_ERR             =31 /* code le plus élevé.
                             /* codes utilisateur plus hauts que
31
} AM_RESULT;
```

Si une procédure AMI renvoie un code utilisateur dépendant de l'application comme résultat, il doit être supérieur à AM_MAX_ERR et inférieur à 256.

NOTE AM_RESULT utilise le même codage que le champ Am_Result transmis dans les paquets.

6.3.10.2.3 Constantes d'adresse

Les constantes figurant dans le Tableau 60 sont des indicatifs réservés.

Tableau 60 – Constantes d'adresse

Constante	Code	Signification
AM_SAME_STATION	0	Cette station, quel que soit son Station_Id
AM_UNKNOWN	255	Station_Id inconnu
AM_MAX_BUSSES	1..16	nombre maximal de couches de liaison pris en charge par cette mise en œuvre
AM_ROUTER_FCT	251	Function_Id d'un Routeur
AM_AGENT_FCT	253	Function_Id d'un Agent
AM_MANAGER_FCT	254	Function_Id d'un Gestionnaire
AM_SAME_NODE	0	communication sur le même nœud qui ne passe pas par le bus de train.
AM_SYSTEM_ADDR	128	bit 0 de la Node_Address indique une System_Address.
AM_ANY_TOPO	0	Topo_Counter est inconnu.

6.3.10.2.4 Type AM_ADDRESS

Un Appelant ou un Répondeur doit identifier l'autre partie par son adresse d'application, qui est du type AM_ADDRESS.

Définition	Type d'une adresse d'application (Appelant ou Répondeur)	

Syntaxe	`typedef struct AM_ADDRESS - représentée poids fort d'abord.`
	`{`
	`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;`	

Éléments	snu	(système, non utilisateur) bit 0 = 0 indique une User_Address bit 0 = 1 indique une System_Address.
	gni	(groupe, non individuel) bit 1 = 0, indique une adresse individuelle (Nœud), bit 1 = 1, indique une adresse de groupe
	node_or_group	Gni = 0, les bits 2..7 sont une Node_Address gni = 1, les bits 2..7 sont une Group_Address
	func_or_stat	si bit 0 du snu = 0, Function_Id si bit 0 du snu = 1, Station_Id
	next_station	Next_Station_Id
	res	réservé, toujours 0
	tpv	(topo valide) indique que le topo_counter suivant est valable
	topo_counter	Topo_Counter ou AM_ANY_TOPO.

NOTE La signification des champs chez l’Appelant et le Répondeur varie comme indiqué ci-dessous.

Un codage convenable d’une adresse d’application est donné dans la Figure 141.

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>snu</td><td>gni</td><td colspan="6">node_or_group</td></tr>
<tr><td colspan="8">func_or_stat</td></tr>
<tr><td colspan="8">next_station</td></tr>
<tr><td>trv</td><td>tpv</td><td colspan="6">topo_counter</td></tr>
</table>

Figure 141 – Codage de AM_ADDRESS

NOTE AM_ADDRESS est un format d’interface, Am_Address est un format de transmission.

6.3.10.3 Côté Appelant

6.3.10.3.1 Identification propre

Un Appelant doit s'identifier par son Function_Id.

NOTE L’Appelant s’identifie dans am_call_request.

6.3.10.3.2 Instances de l’appelant

Etant donné qu’un Appelant peut lancer plusieurs appels avant de recevoir une réponse, la variable caller_ref doit lier la procédure am_call_request à la procédure call_confirm correspondante.

6.3.10.3.3 Système ou Utilisateur (snu)

Si une fonction autre que le Gestionnaire lance un appel avec une System_Address, l’appel ne doit pas être exécuté et une erreur d’adresse doit être signalée dans la procédure call_confirm.

NOTE Une fonction peut appeler une fonction Agent ou Gestionnaire en utilisant son User_Address, en spécifiant next_station, à condition que la communication ne transite pas par le bus de train (node = AM_SAME_NODE).

6.3.10.3.4 Groupe ou individu (gni)

Si l’Appelant attribue la valeur ‘0’ au ‘gni’, le protocole point à point doit être utilisé et les six bits suivants doivent être interprétés comme une Node_Address.

Si l’Appelant attribue la valeur ‘1’ au bit ‘gni’, le protocole de distribution doit être utilisé et les six bits suivants doivent être interprétés comme une Group_Address.

NOTE Le protocole de distribution utilise le même format d’adresse.

6.3.10.3.5 Nœud ou groupe (node_or_group)

Si l’Appelant spécifie (Node_Address <> AM_SAME_NODE), l’appel doit être transféré vers le bus de train.

NOTE Même si la Node_Address du Répondeur est identique à celle de l’Appelant, le message est transféré vers le nœud, qui vérifie le Topo_Counter et renvoie le message sur le réseau de rame.

6.3.10.3.6 Station ou fonction (func_or_stat)

Un Function_Id peut être utilisé avec une User_Address.

Le Gestionnaire peut spécifier (Station_Id = AM_UNKNOWN) dans une System_Address. Cependant, l’appel ne doit pas être exécuté et une erreur d'adresse doit être signalée dans call_confirm si la next_station est AM_UNKNOWN.

NOTE 1 Cela permet à un gestionnaire d'accéder à une station dont le Station_Id est inconnu lors de l’initialisation.

NOTE 2 Lorsqu’un appel est envoyé sur le bus de train, Station_Id = 0 ou 255 adresse le nœud distant quel que soit le Station_Id.

6.3.10.3.7 Next_Station

Next_Station spécifie la Link_Address à laquelle le Message doit être transféré par la suite. Next_Station peut également spécifier la station finale ou une station d'acheminement. Elle doit être calculée comme suit:

a) Si Next_Station est spécifiée (Next_Station_Id <> AM_UNKNOWN), la Link_Address doit être extraite du répertoire de stations, en utilisant Next_Station_Id comme entrée.

b) Si Next_Station n’est pas spécifiée (Next_Station_Id = AM_UNKNOWN), la Link_Address doit être extraite du répertoire de stations, en utilisant l’entrée suivante par défaut:

   - si le message est envoyé au bus de train (Node_Address <> AM_SAME_NODE) ou (distribution), Next_Station_Id doit être extrait du répertoire de fonctions en utilisant la fonction routeur (AM_ROUTER_FCT) comme entrée,
   - si le message n’est pas envoyé sur le bus de train (Node_Address = AM_SAME_NODE),
     – dans le cas d’une System_Address: la valeur de Station_Id doit être attribuée à Next_Station_Id;
     – dans le cas d’une User_Address: Next_Station_Id doit être extrait du répertoire de fonctions en utilisant le Function_Id comme entrée;

c) si (Next_Station_Id = AM_SAME_STATION) ou (Next_Station_Id = this_station), le Messager doit transférer le Call_Message vers un Répondeur local s’il existe.

   Une erreur d’adresse doit être générée si le répertoire de stations ne comporte pas d’entrée pour la Link_Address correspondant au Next_Station_Id.

NOTE Si l’Appelant se trouve sur un nœud, next_station = AM_SAME_STATION.

6.3.10.3.8 Topo_Counter

Le bit 7 (de poids fort) de cet octet doit être à 0.

Le bit 6 de cet octet doit être à 0.

Les bits 0 à 5 doivent contenir un Topo_Counter valide compris entre 1 et 63.

Sinon, tous les bits de cet octet doivent être à 0 (AM_ANY_TOPO).

Une erreur d’adresse doit être générée si l’application spécifie la valeur AM_ANY_TOPO pour tous les appels pour lequel (node <> AM_SAME_NODE).

NOTE Un Appelant ne peut pas envoyer un message point à point sur le bus de train s’il ignore la topographie. Il est prévu de récupérer d’abord la topographie à partir du nœud ou d’une application intermédiaire. Dans le cas d’une configuration fixe du bus de train, toutes les valeurs du Topo_Counter sont admises.

6.3.10.3.9 Utilisation de la Network_Address chez l’Appelant

Une synthèse des modes System_Address et User_Address est donnée dans le Tableau 61.

Tableau 61 – Adresse Système et Adresse Utilisateur

System_Address		**même nœud**	**autre nœud**
Next_Station =	**Station_Id =**	**Link_Address =**	**Link_Address =**
AM_SAME_STATION	AM_SAME_STATION	raccourci vers sa propre station	Node_Address (erreur si this_station n'est pas un nœud)
	<> AM_SAME_STATION et <> AM_UNKNOWN	erreur	
	AM_UNKNOWN	raccourci vers sa propre station	
<> AM_SAME_STATION and <>AM_UNKNOWN	AM_SAME_STATION	stadi (next_station)	stadi (next_station) (l'Agent se trouve sur la station distante)
	<> AM_SAME_STATION et <> AM_UNKNOWN		
	AM_UNKNOWN		
AM_UNKNOWN	AM_SAME_STATION	raccourci vers sa propre station	stadi (fundi(AM_ROUTER_FCT)) (l'Agent se trouve sur le nœud distant)
	<> AM_SAME_STATION et <> AM_UNKNOWN	stadi (Station_Id)	stadi (fundi(AM_ROUTER_FCT)) (l'Agent se trouve sur la station distante)
	AM_UNKNOWN	erreur	stadi (fundi(AM_ROUTER_FCT)) (l'Agent se trouve sur le nœud distant)
User_Address		**même nœud**	**autre nœud**
Next_Station	**Function_Id**	**Link_Address =**	**Link_Address =**
AM_SAME_STATION	quelconque	raccourci vers sa propre station	bus de train, Node_Address (erreur si la station n'est pas un nœud)
<> AM_SAME_STATION and <> AM_UNKNOWN	quelconque	stadi (next_station)	stadi (fundi(AM_ROUTER_FCT))
AM_UNKNOWN	quelconque	stadi (fundi(Function_Id)) (erreur si la fonction n'est pas enregistrée)	stadi (fundi(AM_ROUTER_FCT)) ou (bus de train, Node_Address)

6.3.10.4 Côté Répondeur

6.3.10.4.1 Instances du Répondeur

Les processus du répondeur sont des processus d'application. Plusieurs instances du Répondeur peuvent servir la même fonction en parallèle. L'appelant ne peut pas spécifier l'instance qui sert l'appel.

Chaque fonction du Répondeur doit être liée pour permettre à cette fonction d'appeler la procédure am_receive_request de réception d'un appel entrant et la procédure am_reply_request de réponse à un appel reçu.

Les processus du Répondeur ne sont pas bloqués en attendant Call_Message ou pendant le transfert d'un Reply_Message. Ils sont informés qu'un Call_Message a été reçu ou que le transfert du Reply_Message est terminé.

Les procédures de confirmation à appeler pour notification sont spécifiées dans la procédure de lien. Elles sont donc identiques pour toutes les instances de la même fonction du Répondeur.

Une instance du Répondeur doit répondre à chaque appel reçu ou l'annuler avant de pouvoir émettre une am_receive_request supplémentaire. Chaque demande qui n'a pas été confirmée peut également être annulée. Une demande qui a été annulée n'est pas confirmée.

6.3.10.4.2 Identification du Répondeur

Etant donné qu'une fonction peut être exécutée par plusieurs instances, la variable replier_ref doit relier am_receive_request aux receive_confirm, am_reply_request et reply_confirm correspondantes.

Une instance du Répondeur doit être identifiée par son Function_Id et son External_Reference.

NOTE La Couche Session conserve l'adresse complète du Répondeur reçue dans le Call_Message pour le Reply_Message. Elle conserve également External_Reference comme partie du Conversation_Id.

6.3.10.4.3 Système ou utilisateur (snu)

La valeur '1' doit être attribuée au bit 'snu' si le message a été reçu par une System_Address. Dans ce cas, la fonction Agent doit être appelée, l'Appelant implicite étant le Gestionnaire.

La valeur '0' doit être attribuée au bit 'snu' si le message a été reçu avec une User_Address.

NOTE Une autre fonction peut s'adresser à l'Agent par une User_Address, mais uniquement à partir des stations reliées au même nœud.

6.3.10.4.4 Groupe ou individu (gni)

La valeur 1 ou 0 doit être attribuée au bit 'gni' pour indiquer, respectivement, que le message a été reçu sur une adresse de diffusion ou sur une adresse point à point.

NOTE Cela permet d'appeler indifféremment un Répondeur en utilisant le protocole point à point ou de diffusion.

6.3.10.4.5 Nœud ou Groupe

Si une adresse individuelle ou une Group_Address est utilisée, les six bits suivants doivent indiquer la Node_Address de l'Appelant, ou AM_SAME_NODE si l'Appelant a spécifié AM_SAME_NODE dans son adresse du Répondeur.

NOTE Si l'Appelant spécifie la Node_Address, cette adresse est envoyée au Répondeur même si le message ne passe pas par le bus de train.

6.3.10.4.6 Next_Station

Next_Station doit être le Station_Id de la station sur laquelle l'appel a été reçu ou, si la station finale est le nœud lui-même, Next_Station doit être AM_SAME_STATION.

6.3.10.4.7 Topo_Counter

Le Répondeur doit recevoir le Topo_Counter du nœud auquel il est relié si le message a été transféré sur le nœud. Sinon, ce champ est mis à AM_ANY_TOPO.

NOTE Le Répondeur est responsable d'assurer que la valeur du Topo_Counter d'un Call_Message correspond à la valeur de my_topo.

6.3.10.5 Initialisation

Le Service de Messagerie est initialisé à différents niveaux par les procédures ci-dessous.

Les paragraphes qui suivent n'impliquent pas une mise en œuvre particulière. Toute interface qui fournit la même sémantique est autorisée.

6.3.10.5.1 Procédure am_init

Définition	Initialise le Messager et appelle am_clear_dir pour initialiser le répertoire.	
Syntaxe	`AM_RESULT`	`am_init (void);`
Résultat	AM_RESULT	AM_OK, AM_FAILURE
Usage	Cette procédure doit être appelée lors de l'initialisation du dispositif avant d'appeler une autre procédure am_xxx.	

6.3.10.5.2 Procédure am_announce_device

Définition	Annonce la configuration du dispositif	

Syntaxe	`AM_RESULT`
	`UNSIGNED16`
	`UNSIGNED16`
	`UNSIGNED16`
	`UNSIGNED8`
		`am_announce_device`
		`(`
		`max_call_number,`
		`max_inst_number,`
		`default_reply_timeout,`
		`my_credit`
		`);`

Entrée	max_call_number	nombre d'appels simultanés sur ce dispositif.
	max_inst_number	nombre d'instances simultanées pour un Répondeur sur ce dispositif (trois par défaut)
	default_reply_tmo	temporisation de réponse par défaut pour les demandes d'appel
	my_credit	crédit maximal (accepté) pour toutes les connexions aboutissant sur ce dispositif. Il est abrégé à AM_MAX_CREDIT.
Renvoi		toute valeur de AM_RESULT
Usage	Cette procédure obtient la Node_Address directement de la couche de liaison.	

6.3.10.5.3 Procédure am_show_busses

Définition	Récupère le nombre de couches de liaison (bus) reliées à cette station et précise leur Bus_Id	

Syntaxe	`AM_RESULT am_show_busses`
	`(`
	`UNSIGNED8 * nr_of_busses,`
	`UNSIGNED8 link_id_list [AM_MAX_BUSSES]`
	`);`	

Renvoi		toute valeur de AM_RESULT
Sortie	nr_of_busses	nombre de couches de liaison reliées (et nombre d'éléments dans link_id_list)
	link_id_list	liste des couches de liaison, avec au moins pour chacune le Bus_Id.

6.3.10.5.4 Procédure am_set_current_tc

Définition	Attribue la valeur en cours this_topo à Topo_Counter pour la couche réseau.	

Syntaxe	`AM_RESULT am_set_current_tc`
	`(`
	`UNSIGNED8 this_topo`
	`);`	

Entrée	this_topo	Topo_Counter ou AM_ANY_TOPO si le Topo_Counter est inconnu.
Renvoi		AM_OK si (AM_ANY_TOPO ≤ this_topo < 63) AM_FAILURE sinon.
Usage	1 – Une Application (Appelant ou Répondeur) récupère la valeur en cours du Topo_Counter et la copie dans sa variable my_topo. En général, cette valeur varie d'une application à l'autre, même à l'intérieur du même nœud, puisqu'une inauguration peut avoir lieu à tout moment et que les applications ne sont pas censées être informées de chaque changement. 2 – La manière selon laquelle l'application reçoit le Topo_Counter n'est pas spécifiée. Elle peut le faire par des messages de gestion, par un accès direct à la couche de liaison sur un nœud, par une variable périodique, etc. 3 – Si this_topo n'est pas égal à AM_ANY_TOPO, le Messager rejettera les appels suivants faits avec une valeur de Topo_Counter qui n'est pas égale à celle-ci ou à AM_ANY_TOPO, et avec le résultat AM_INAUG_ERR dans 'call_confirm'. 4 – La valeur initiale de this_topo est AM_ANY_TOPO.	

6.3.10.6 Interface du répertoire de stations

Le répertoire de stations est facultatif. Les dispositifs simples peuvent procéder à la mise en correspondance de manière fixe. Lorsqu'un répertoire de stations est utilisé, il est rendu disponible par les procédures suivantes.

Les paragraphes qui suivent n'impliquent pas une mise en œuvre particulière. Toute interface qui fournit la même sémantique est autorisée.

6.3.10.6.1 Type AM_STADI_ENTRY

Définition	Type d'entrée d'un répertoire de stations	

Syntaxe	`typedef struct`
	`{`
	`UNSIGNED8 station;`
	`UNSIGNED8 next_station;`
	`ENUM8 bus_id;`
	`UNSIGNED64 device_adr;`
	`} AM_STADI_ENTRY;`	

Éléments	station	Station_Id (clé pour récupérer next_station)
	next_station	Next_Station Next_Station est égale à Station_Id pour une station qui peut être atteinte directement
	bus_id	Bus_Id
	device_adr	Device_Address (dépendante du bus)

6.3.10.6.2 Procédure am_stadi_write

Définition	Insère un certain nombre d'entrées dans le répertoire de stations, la validité et la cohérence de chacune d'elles étant vérifiées, et l'entrée pouvant être rejetée.	

Syntaxe	`AM_RESULT am_stadi_write`
	`(`
	`const entries[],`
	`AM_STADI_ENTRY nr_of_entries`
	`UNSIGNED8 );`	

Entrée	entries	liste des nouvelles entrées du répertoire de stations
	nr_of_entries	nombre d'éléments dans les entrées.
Renvoi		AM_OK: toutes les entrées peuvent être écrites avec succès dans le répertoire de stations. AM_FAILURE: une entrée de la liste n'est pas satisfaisante. Dans ce cas, seules ces entrées sont rejetées.

6.3.10.6.3 Procédure am_stadi_read

Définition	Lit un certain nombre d'entrées dans le répertoire de stations.	

Syntaxe	`AM_RESULT am_stadi_read`
	`(`
	`AM_STADI_ENTRY entries[],`
	`UNSIGNED8 nr_of_entries`
	`);`	

Entrée	entries[].station	entrées à lire.
	nr_of_entries	nombre d'entrées.
Renvoi		AM_OK, AM_FAILURE
Sortie	entries[].next_station	les champs entries[].next_station contiennent les informations de sortie.

6.3.10.7 Interface du répertoire de fonctions

Les paragraphes qui suivent n'impliquent pas une mise en œuvre particulière. Toute interface qui fournit la même sémantique est autorisée.

6.3.10.7.1 Type AM_DIR_ENTRY

Définition	Type d'entrée d'un répertoire de fonctions.

Syntaxe	`typedef struct AM_DIR_ENTRY {`
	`UNSIGNED8 function;`
	`UNSIGNED8 station;`
	`} AM_DIR_ENTRY;`

6.3.10.7.2 Procédure am_clear_dir

Définition	Attribue la valeur AM_UNKNOWN au Station_Id de toutes les fonctions du répertoire.	
Syntaxe	`AM_RESULT am_clear_dir (void);`	
Sortie		AM_OK, AM_FAILURE

6.3.10.7.3 Procédure am_insert_dir_entries

Définition	Insère la Station_Id de chaque Function_Id figurant dans les premiers éléments 'number_of_entries' de la liste function_list.	

Syntaxe	`AM_RESULT am_insert_dir_entries (`
	`AM_DIR_ENTRY * function_list,`
	`unsigned number_of_entries`
	`);`	

Entrée	function_list	liste du répertoire de fonctions
	number_of_entries	nombre d'éléments dans cette liste
Renvoi		AM_OK, AM_FAILURE

6.3.10.7.4 Procédure am_remove_dir_entries

Définition	Attribue la valeur AM_UNKNOWN au Station_Id pour chaque Fonction_Id figurant dans les premiers éléments 'number_of_entries' de la liste des fonctions.	
Syntaxe	`AM_RESULT am_remove_dir_entries ( AM_DIR_ENTRY * function_list, unsigned number_of_entries );`	
Entrée	function_list	liste du répertoire de fonctions
	number_of_entries	nombre d'éléments dans cette liste
Renvoi		AM_OK, AM_FAILURE

6.3.10.7.5 Procédure am_get_dir_entry

Définition	Lit le Station_Id d'un Fonction_Id donné dans le répertoire de fonctions.	
Syntaxe	`AM_RESULT am_get_dir_entry ( UNSIGNED8 function, UNSIGNED8 * station );`	
Entrée	function	Function_Id (clé)
Sortie	station	Station_Id de la station sur laquelle la fonction est exécutée, ou AM_UNKNOWN si aucune station n'est attribuée à la fonction.
Renvoi		AM_OK, AM_FAILURE

6.3.10.8 Interface du répertoire de groupes

Les paragraphes qui suivent n'impliquent pas une mise en œuvre particulière. Toute interface qui fournit la même sémantique est autorisée.

6.3.10.8.1 Type AM_GROUP

Définition	Type d'une entrée dans le répertoire de groupes
Syntaxe	`typedef UNSIGNED8 AM_GROUP;`
Usage	Les groupes sont identifiés par une adresse à 6 bits, représentée par un octet, les deux bits de poids fort étant ignorés.

6.3.10.8.2 Procédure am_clear_groups

Définition	Efface toutes les entrées du répertoire de groupes.	
Syntaxe	`AM_RESULT`	`am_clear_groups (void);`
Renvoi		AM_OK, AM_FAILURE

6.3.10.8.3 Procédure am_insert_member

Définition	Enregistre ce Station_Id comme un membre du répertoire de groupes.	

Syntaxe	`AM_RESULT`
	
	`AM_GROUP`
		`am_insert_member`
		`(`
		`group`
		`);`

Entrée	group	Groupe auquel cette station doit appartenir.
Renvoi		AM_OK, AM_FAILURE

6.3.10.8.4 Procédure am_remove_member

Définition	Enlève ce Station_Id comme membre d'un Groupe.	

Syntaxe	`AM_RESULT`
	
	`AM_GROUP`
		`am_remove_member`
		`(`
		`group`
		`);`

Entrée	group	Groupe duquel cette station doit être enlevée.
Renvoi		AM_OK, AM_FAILURE

6.3.10.8.5 Procédure am_member

Définition	Vérifie si le Station_Id indiqué est membre d'un Groupe.	

Syntaxe	`AM_RESULT`
	
	`AM_GROUP`
		`am_member`
		`(`
		`group`
		`);`

Entrée	group	Groupe auquel ce Station_Id appartient.
Renvoi		AM_OK si le Station_Id est membre du Groupe, sinon, AM_FAILURE.

6.3.10.9 Interface d'application de l'Appelant

Les paragraphes qui suivent n'impliquent pas une mise en œuvre particulière. Toute interface qui fournit la même sémantique est autorisée.

6.3.10.9.1 Procédure am_call_request

Définition	Demande l'envoi d'un Call_Message, configure les structures de données pour recevoir le Reply_Message et souscrit les procédures d'indication.	

Syntaxe	`void`
	`UNSIGNED8`
	`const AM_ADDRESS`
	`*`
	`void *`
	`UNSIGNED32`
	`void *`
	`UNSIGNED32`
	`UNSIGNED16`
	`AM_CALL_CONFIRM`
	`void *`
		`am_call_request`
		`(`
		`caller_function,`
		`replier,`
		`out_msg_adr,`
		`out_msg_size,`
		`in_msg_adr,`
		`in_msg_size,`
		`reply_timeout,`
		`call_confirm,`
		`caller_ref`
		`);`

Entrée	caller_function	Function_Id de l'Appelant. AM_MANAGER_FCT doit être utilisé si l'adresse du Répondeur est une System_Address
	replier	Adresse d'application de la fonction ou station du Répondeur.
	out_msg_adr	pointeur vers le Call_Message à transmettre
	out_msg_size	longueur totale du Call_Message en octets
	in_msg_adr	pointeur vers le tampon contenant le Reply_Message.
	in_msg_size	taille maximale en octets du Reply_Message accepté
	reply_timeout	valeur de la temporisation en multiples de 64 ms pour la réponse suivant le transfert du Call_Message.
	call_confirm	pointeur vers la procédure de confirmation d'appel. 'call_confirm' est appelée sauf si 'am_call_request' est annulée avec succès par am_call_cancel.
	caller_ref	External_Reference de l'appel à renvoyer par 'call_confirm'. Elle peut être utilisée par l'Appelant.
Renvoi		Cette procédure ne renvoie aucune valeur, le résultat étant fourni par 'call_confirm'.
Usage	1 – Les procédures am_init et am_announce_device doivent être appelées avant 'am_call_request'. 2 – Si in_msg_adr est NULL, le Messager attribue un tampon au Reply_Message. L'Appelant est alors chargé de renvoyer ce tampon après utilisation en appelant am_buffer_free. 3 – L'appel est rejeté si une conversation avec les mêmes adresses Appelant et Répondeur (et dans le même sens) a déjà été établie. 4 – Une entrée de répertoire de fonctions pour la fonction du Répondeur doit être définie si le Station_Id du Répondeur est AM_UNKNOWN. 5 – Aucune conversation n'a lieu sur le bus si l'Appelant et le Répondeur se trouvent dans la même station. 6 – Un appel avec une System_Address est rejeté si le Function_Id n'est pas 254 (Gestionnaire). 7 – L'Appelant ne peut modifier les tampons de message tant que call_confirm n'est pas appelé. 8 – La temporisation de réponse par défaut spécifiée avec am_announce_device est attendue si la temporisation de réponse est 0.	

6.3.10.9.2 Type AM_CALL_CONFIRM

Définition	Lorsqu'un appel demandé s'achève, renvoyant un état d'erreur ou le Reply_Message reçu, la Couche Session doit appeler la procédure 'call_confirm' de l'Appelant, qui est de ce type.	

Syntaxe	`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`
	`);`	

Entrée	caller_function	Function_Id de l'Appelant
	replier	Adresse d'application de la fonction ou station du Répondeur.
	am_caller_ref	valeur renvoyée qui a été spécifiée dans la 'am_call_request' associée.
	in_msg_adr	pointeur vers un tampon contenant le Reply_Message reçu. Il est NULL si l'Appelant n'a pas fourni un tampon pour le Reply_Message et si, en même temps, la valeur 0 a été attribuée à in_msg_size.
	in_msg_size	taille totale en octets du Reply_Message. Il est 0 en cas d'erreur ou si l'application Répondeur ne renvoie qu'un état.
	status	génère un code erreur < AM_MAX_ERR ou, en cas de réussite, l'état fourni par le Répondeur.
Usage	1 – La procédure 'call_confirm' doit être préalablement souscrite par 'am_call_request'. 2 – AM_MANAGER_FCT est renvoyée si l'adresse Répondeur est une System_Address. 3 – La confirmation d'appel renvoie le tampon Call_Message à l'Appelant de manière implicite. 4 – Un tampon Reply_Message attribué par le Messager doit être renvoyé avec am_buffer_free après son utilisation.	

6.3.10.9.3 Procédure 'am_call_cancel'

Définition	Annule une demande d'appel qui n'a pas encore été confirmée.	

Syntaxe	`AM_RESULT am_call_cancel (`
	`UNSIGNED8 caller_function,`
	`const AM_ADDRESS * replier`
	`);`	

Entrée	caller_function	Function_Id de l'Appelant
	replier	Adresse d'application de la fonction ou de la station appelée.
Renvoi		AM_OK annulation réussie, AM_FAILURE pour les autres erreurs
Usage	La procédure de confirmation d'appel n'est pas appelée si la valeur renvoyée est AM_RESULT.	

6.3.10.10 Interface d'application du Répondeur

Les paragraphes qui suivent n'impliquent pas une mise en œuvre particulière. Toute interface qui fournit la même sémantique est autorisée.

6.3.10.10.1 Procédure 'am_bind_replier'

Définition	Informe le Messager de la présence d'un Répondeur et relie ses procédures.	
Syntaxe	AM_RESULT UNSIGNED8 AM_RECEIVE_CONFIRM AM_REPLY_CONFIRM	am_bind_replier (replier_function, receive_confirm, reply_confirm);
Entrée	replier_function	Function_Id du Répondeur à lier
	receive_confirm	procédure de confirmation de réception appelée à la fin d'une demande de réception
	reply_confirm	procédure de confirmation de réponse appelée à la fin d'une réponse.
Renvoi		AM_OK = 0 le lien a réussi, sinon am_bind_replier n'a pas d'effet AM_ALREADY_USED cette fonction du Répondeur est déjà liée, les procédures de confirmation ne sont pas modifiées AM_NO_LOC_MEM_ERR pas de mémoire pour le tableau de liens, aucune fonction du Répondeur ne peut être liée AM_FAILURE le tableau de liens est saturé. La taille du tableau de liens est définie par am_announce_device.
Usage	1 – 'am_init' et 'am_announce_device' doivent être appelés avant 'am_bind_replier'. 2 – Chaque Répondeur doit être lié avant de pouvoir émettre une 'am_receive_request'.	

6.3.10.10.2 Procédure 'am_unbind_replier'

Définition	Annule toutes les instances du Répondeur spécifié et libère les liens.	
Syntaxe	AM_RESULT UNSIGNED8	am_unbind_replier (replier_function);
Entrée	replier_function	Function_Id du Répondeur à délier
Renvoi		AM_OK, AM_FAILURE
Usage	Les appels qui ont été reçus avant d'appeler am_unbind_replier, mais qui n'ont pas eu de réponse sont annulés.	

6.3.10.10.3 Procédure ‘am_receive_request’

Définition	Signale qu’un Répondeur est prêt à recevoir un appel entrant.	

Syntaxe	`AM_RESULT am_receive_request`
	`(`
	`UNSIGNED8 replier_function,`
	`void * in_msg_adr,`
	`UNSIGNED32 in_msg_size,`
	`void * replier_ref`
	`);`	

Entrée	replier_function	Function_Id du Répondeur qui attend un appel. Function_Id = 253 implique une System_Address.
	in_msg_adr	Pointeur vers un tampon pour le Call_Message entrant. Le Messager attribue un tampon au Call_Message si in_msg_adr est NULL. Ce tampon ne peut pas être modifié tant que am_receive_request n’est pas confirmée ou annulée.
	in_msg_size	Taille totale maximale en octets du Call_Message qui peut être acceptée.
	replier_ref	Référence externe renvoyée avec la procédure receive_confirm associée. C'est en même temps une référence d’instance qui fait la distinction entre les instances du Répondeur qui servent le même Répondeur
Renvoi		AM_OK la procédure liée de confirmation de réception est appelée pour transmettre un appel reçu si la demande n’est pas annulée. AM_ALREADY_USED la même instance du Répondeur a déjà émis une demande de réception qui n’a été ni confirmée, ni annulée ni répondue. AM_FAILURE Le Répondeur n’est pas lié; AM_NO_LOC_MEM_ERR la mémoire n’est pas suffisante pour accepter une am_receive_request; AM_NR_OF_INST_OVF le nombre d’am_receive_request simultanées émises dépasse la limite imposée par le paramètre max_inst_number d’am_announce_device.
Usage	Cette procédure demande que la procédure ‘am_bind_replier’ pour le même Répondeur ait déjà été appelée.	

6.3.10.10.4 Type AM_RECEIVE_CONFIRM

Définition	Lorsque la Couche Session reçoit un Call_Message, elle doit appeler la procédure de confirmation de réception receive_confirm, qui est de ce type.	

Syntaxe	`typedef void ( * AM_RECEIVE_CONFIRM )`
	`(`
	`UNSIGNED8 replier_function,`
	`const AM_ADDRESS caller,`
	`* in_msg_adr ,`
	`void * in_msg_size,`
	`UNSIGNED32 replier_ref`
	`void * );`	

Entrée	replier_function	Function_Id du Répondeur spécifié dans la ‘am_receive_request’ correspondante
	caller	Adresse de l’Appelant
	in_msg_adr	Pointeur vers un tampon qui contient le Call_Message.
	in_msg_size	Taille totale en octets du Call_Message reçu.
	replier_ref	Référence externe telle que spécifiée dans la ‘am_receive_request’ correspondante
Usage	1 - La procédure de confirmation de réception ‘receive_confirm’ a déjà été souscrite par am_bind_replier. 2 - Si l’instance du Répondeur n’a pas fourni un tampon dans ‘am_receive_request’, le tampon de réponse est fourni par le Messager et le Répondeur doit le renvoyer après utilisation avec ‘am_buffer_free’.	

6.3.10.10.5 Procédure 'am_reply_request'

Définition	Demandes d'envoi d'un Reply_Message en réponse à un Call_Message déjà reçu.	

Syntaxe	`AM_RESULT am_reply_request`
	`(`
	`UNSIGNED8 replier_function,`
	`void * out_msg_adr,`
	`UNSIGNED32 out_msg_size,`
	`void * replier_ref`
	`AM_RESULT status);`	

Entrée	replier_function	Function_Id du Répondeur spécifié dans la 'am_receive_request' correspondante
	replier_ref	Référence externe spécifiée dans la 'am_receive_request' correspondante
	out_msg_adr	pointeur vers le tampon Reply_Message. Ce tampon ne doit pas être modifié avant la confirmation de la demande de réponse. Si out_msg_adr est NULL, seul l'état est transmis à l'Appelant.
	out_msg_size	taille totale en octets du Reply_Message.
	status	résultat de l'exécution de l'appel fourni par le Répondeur, transmis à l'Appelant en plus du Reply_Message lui-même.
Renvoi		AM_OK, AM_FAILURE
Usage	1 – Chaque appel reçu doit faire l'objet d'une réponse avec 'am_reply_request' ou d'une annulation avec 'am_receive_cancel'. 2 – Cette procédure est renvoyée avant la transmission du Reply_Message. 3 – Le Messager retransmet ce Reply_Message et l'état spécifié à l'Appelant. 4 – L'adresse de l'Appelant est récupérée en interne de l'instance du Répondeur.	

6.3.10.10.6 Type 'AM_REPLY_CONFIRM'

Définition	Lorsque la totalité d'un Reply_Message a été envoyée et que l'Appelant a accusé réception, ou si une erreur se produit, la Couche Session appelle la procédure de confirmation de réponse 'reply_confirm' qui est de ce type.	

Syntaxe	`typedef void ( * AM_REPLY_CONFIRM )`
	`(`
	`UNSIGNED8 replier_function,`
	`void * replier_ref`
	`);`	

Entrée	replier_function	Function_Id du Répondeur spécifié dans la 'am_receive_request' correspondante
	replier_ref	Référence externe spécifiée dans la 'am_receive_request' correspondante
Usage	1 – 'reply_confirm' a déjà été souscrite par 'am_bind_replier'. 2 – Cette procédure renvoie le tampon Reply_Message à l'instance du Répondeur. 3 – La confirmation de réponse autorise une 'am_receive_request' supplémentaire pour la même instance du Répondeur.	

6.3.10.10.7 Procédure ‘am_receive_cancel’

Définition	Annule une ‘am_receive_request’ ou une ‘am_reply_request’ qui n’a pas été confirmée ou annonce qu’un appel reçu n’aura pas de réponse.	
Syntaxe	`AM_RESULT am_receive_cancel ( UNSIGNED8 replier_function, void * replier_ref );`	
Entrée	replier_function	Function_Id du Répondeur spécifié dans la ‘am_receive_request’ correspondante
	replier_ref	Référence externe spécifiée dans la ‘am_receive_request’ correspondante
Renvoi		AM_OK, AM_FAILURE
Usage	1 – La procédure de confirmation d’une ‘reply request’ annulée ne sera plus appelée. 2 – L’utilisateur doit déterminer si la demande de réception est achevée (c'est-à-dire si un Call_Message a été reçu) ou non et libérer les tampons dynamiques pour les Call_Messages.	

6.3.10.11 Procédure ‘am_buffer_free’

Définition	Libère un tampon de message déjà attribué par la Couche Session après utilisation.	
Syntaxe	`AM_RESULT am_buffer_free ( void * in_msg_adr, UNSIGNED32 size );`	
Entrée	in_msg_adr	pointeur vers le tampon libéré.
	size	taille totale de ce tampon en octets.
Renvoi		AM_OK, AM_FAILURE
Usage	L’attribution des tampons est indépendante de la gestion des groupements de paquets.	

6.3.10.12 Interface d’application de distribution

L’interface d’application des messages de distribution doit être identique à celle des messages point à point.

Le répondeur n’est pas censé renvoyer un Reply_Message, mais doit appeler ‘am_reply_request’ pour libérer un tampon dynamique, le cas échéant. Cependant, dans ce cas, aucun Reply_Message ne doit être généré.

6.4 Présentation et codage des données transmises ou stockées

6.4.1 Objet

Le présent paragraphe spécifie des types de données, et définit une notation syntaxique abstraite permettant d’exprimer ces types de données et les règles de codage utilisées pour leur transmission ou leur stockage. Cette notation repose sur ASN.1 (ISO/CEI 8824). Elle contient des constructions supplémentaires adaptées à la communication en temps réel. En revanche, les interfaces internes d'un dispositif sont spécifiées en langage C, qui est moins précis.

6.4.2 Organisation des données

6.4.2.1 Format de transmission

La présente norme spécifie l'ordre de transmission des bits et des mots sur le Réseau Embarqué de Train. Elle définit pour cela un certain nombre de types primitifs et structurés. La signification des données n'entre pas dans le domaine d'application de cette norme.

6.4.2.2 Format du Traffic_Store

La présente norme préconise de stocker les données dans le Traffic_Store au même format que celui de leur transmission sur le bus, en les traitant comme un tableau d'octets.

6.4.2.3 Format des données d'application

La présente norme ne spécifie pas le format des données d'application. Les procédures d'interface sont censées convertir les formats de données de l'application vers ceux qui sont utilisés pour le stockage ou la transmission, et inversement.

6.4.2.4 Règles générales

a) Les structures de données doivent être numérotées de gauche à droite et de haut en bas, dans le sens de lecture d'un texte français. Le décalage du premier élément (en haut, à gauche) est nul.

b) Une mémoire doit être considérée comme un tableau d'octets transmis en ordre croissant d'adresse, quelle que soit la taille des blocs transmis (par octets, par mots de 32 bits, etc.). Le décalage du premier octet est nul.

c) Les bits à l'intérieur d'une structure de données doivent être identifiés par leur décalage par rapport au début de la structure. Si cette structure contenait un nombre entier, le bit de poids faible serait le bit avec un décalage nul. Ce bit est dit être 'le bit le plus à gauche' de la structure de données.

d) Pour améliorer la compréhension, la numérotation de bit dans les articles suivants 6.4.3.x est alignée à une vue des programmeurs par rapport à la structure de données de la mémoire de trafic. Un décalage de bit x peut donc être interprété comme la puissance $x^{\text{ième}}$ de deux 2^x. Le $x^{\text{ième}}$ bit d'un bitset peut être calculé comme un décalage d'octet et de bit comme suit: décalage d'octet = x divisé par 8 arrondi à l'élément inférieur, décalage de bit (à l'intérieur de l'octet) = x modulo 8.

e) Toutes les données doivent être transmises en commençant par l'octet de ppoids fort (Big-Endian).

f) L'ordre de transmission des bits d'un octet est considéré comme une particularité du bus, invisible pour le programmeur. En particulier, le protocole HDLC tel que l'utilise le WTB transmet le bit de poids faible d'un octet (décalage 7) en premier, alors que le MVB le transmet en dernier.

g) Les informations relatives au type des données ne sont pas envoyées avec les données. Ces types doivent être définis et approuvés à l'avance par les utilisateurs du TNC pour une application donnée.

h) Les éléments d'un type structuré (Enregistrement, Séquence) doivent être transmis dans l'ordre de leur déclaration.

i) Les tableaux doivent être transmis en ordre croissant d'indice. Les tableaux à dimensions multiples sont transmis dans l'ordre d'apparition de leurs indices (par exemple, ARRAY OF [ligne, colonne] est transmis ligne par ligne).

j) Pour faciliter la mise en œuvre, une variable doit être stockée à une position dont le décalage est un multiple de sa taille (alignement).

k) Les données de longueur variable (tableaux ouverts, enregistrements, ensembles, etc.) ne doivent pas être utilisées comme Process_Variables, mais peuvent être transmises dans des messages.

6.4.2.5 Relation avec ASN.1

L'ISO/CEI 8824 définit la syntaxe abstraite numéro un (ASN.1) pour spécifier des données sous une forme lisible automatiquement.

Bien que ASN.1 n'impose pas de syntaxe de transfert, ce langage ne peut exprimer les règles de codage compact fréquemment utilisées par les programmeurs quand la bande passante ou le temps sont limités. ASN.1 ne peut pas non plus exprimer les codages existants qui ne respectent pas sa méthode de structure.

En conséquence, la présente norme définit une syntaxe abstraite reposant sur ASN.1, qui exprime également le codage des données, et qui permet de spécifier bit par bit le contenu des données.

Les mots-clés suivants ont été ajoutés à 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#

Cette notation utilise des règles de codage compact:

- elle suppose que tous les types définis par l'utilisateur sont reconnus par le destinataire;
- elle utilise des types primitifs de taille fixe (dans ASN.1, un entier peut avoir une taille quelconque);
- elle inclut explicitement la taille des éléments dans un champ prévu à cet effet, si nécessaire;
- elle utilise des mots clés différents d'ASN.1 lorsqu'il existe une possibilité de confusion sur la sémantique (par exemple ONE_OF au lieu de CHOICE, SOME_OF au lieu de SET);
- elle n'utilise pas d'étiquette de type implicite, sauf pour les types ONE_OF et SOME_OF où l'étiquettage est fait explicitement par un champ dédié;
- elle n'a pas de champs optionnels (sauf dans SOME_OF);
- elle n'est pas alignée, bien qu'il soit possible de spécifier l'alignement.

Les règles suivantes sont utilisées:

- les mots-clés, y compris les types de base, ainsi que les constantes sont entièrement en majuscule;
- les indicatifs de type commencent par une lettre majuscule;
- les indicatifs de champs commencent par une lettre minuscule.

6.4.3 Notation des types primitifs

6.4.3.1 Notation pour le type booléen

6.4.3.1.1 Définition

Type primitif ayant deux valeurs distinctes, TRUE et FALSE.

NOTE 1 Ceci est également la définition en ASN.1 de BooleanType.

NOTE 2 Ce type est utilisé pour représenter des entrées et sorties binaires (relais, voyant, micro-contact, etc.).

6.4.3.1.2 Syntaxe

```
BooleanType::= BOOLEAN1
```

6.4.3.1.3 Codage

Une variable de type booléen doit être codée sur un bit:

1er	Interprétation
0	FALSE
1	TRUE

6.4.3.2 Notation pour le type antivalent

6.4.3.2.1 Définition

Type primitif ayant quatre valeurs distinctes.

NOTE 1 Il ne s'agit pas d'un type ASN.1.

NOTE 2 Ce type est utilisé pour les variables de contrôle, pour d'autres variables ou pour des booléens critiques.

6.4.3.2.2 Syntaxe

```
AntivalentType::= ANTIVALENT2
```

6.4.3.2.3 Codage

Une variable de type antivalent doit être codée sur 2 bits, le premier correspondant à la valeur booléenne de la variable et le second à son inverse.

Ceci définit quatre états, comme suit:

1	0	Interprétation
2^{+1}	2^{0}	
0	0	ERROR
0	1	FALSE
1	0	TRUE
1	1	UNDEFINED

NOTE Les états ERROR et UNDEFINED peuvent être interprétés comme des états normaux par une application.

6.4.3.3 Notation des types entiers non signés

6.4.3.3.1 Définition

Type primitif ayant des valeurs distinctes qui sont des nombres entiers positifs, y compris le zéro (comme valeur unique) et ayant une taille fixe définie par le suffixe #.

NOTE Il s'agit du type IntegerType en ASN.1, restreint à une taille fixe (#) et à des valeurs non négatives.

6.4.3.3.2 Syntaxe

```
UnsignedType::= UNSIGNED#, (# = entier positif sans signe).
```

6.4.3.3.3 Codage

Un entier non signé doit être transmis en représentation binaire, le bit de poids fort en premier.

Quand la taille de la valeur transmise est plus petite que le type UNSIGNED#, cette valeur doit être justifiée à droite et étendue vers la gauche par des zéros.

6.4.3.3.3.1 Codage de UNSIGNED8

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Plage: 0..255

6.4.3.3.3.2 Codage de UNSIGNED16

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

Plage: 0..65535

6.4.3.3.3.3 Codage de UNSIGNED32

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

Plage: 0..+2^{32}–1

6.4.3.4 Notation du type entier

6.4.3.4.1 Définition

Type primitif aux valeurs distinctes qui sont des nombres entiers positifs et négatifs, incluant le zéro (comme une valeur unique), dont la taille fixe en bits est définie par le suffixe #.

NOTE Il s'agit du type 'integer type' en ASN.1, restreint à une taille fixe de #.

6.4.3.4.2 Syntaxe

```
IntegerType::= INTEGER#, (# = any unsigned integer).
```

6.4.3.4.3 Codage

La valeur doit être transmise en représentation binaire en complément à deux, le premier bit transmis étant celui du signe.

Si la taille de la valeur transmise est plus petite que le type INTEGER#, elle doit être justifiée à droite et étendue à gauche par un signe (1 si la valeur est négative et 0 dans le cas contraire).

6.4.3.4.3.1 Codage de INTEGER8

7	6	5	4	3	2	1	0
signe	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Plage: –128 .. +127

EXEMPLE ‘1111 1110’B = – 2

6.4.3.4.3.2 Codage de INTEGER16

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

Plage: -2^{15}.. $2^{15} - 1$

6.4.3.4.3.3 Codage de INTEGER32

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

Plage: -2^{31}..$+2^{31} - 1$

6.4.3.5 Notation du type énuméré

6.4.3.5.1 Définition

Type primitif dont les valeurs sont des indicatifs distincts donnés dans le cadre de la notation du type, ayant une taille fixe en bits définie par le suffixe #.

NOTE Il s’agit d’un type ENUMERATED d’ASN.1, restreint à une taille fixe de #.

6.4.3.5.2 Syntaxe

```
EnumeratedType::= ENUM#{Énumération}
```

avec

```
(# = entier positif sans signe)
Enumeration::= NamedNumber | Enumeration, NamedNumber
```

et

```
NamedNumber::= identifier (UnsignedNumber) | identifier (DefinedValue)
```

Les valeurs peuvent être listées dans un ordre quelconque.

EXEMPLE

```
Day_Of_Week_Type::= ENUM4
  {
  LUNDI          (1),
  MARDI          (2),
  MERCREDI       (3),
  JEUDI          (4),
  VENDREDI       (5),
  SAMEDI         (6),
  DIMANCHE       (7),
  INDÉFINI       (0)
  }
```

La valeur ‘2’ signifie ‘MARDI’.

6.4.3.5.3 Codage

Les valeurs de ENUM# doivent être représentées par un entier non signé occupant la même place.

6.4.3.5.3.1 Codage de ENUM4

3	2	1	0
2^3	2^2	2^1	2^0

Plage: 0..15

EXEMPLE: ‘0001’B signifie ‘LUNDI’ dans l’exemple précédent.

6.4.3.5.3.2 Codage de ENUM8

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Plage: 0..255

EXEMPLE: ‘0000 0001’B signifie ‘LUNDI’ dans l’exemple précédent (en le considérant comme ENUM8 plutôt qu’ENUM4).

6.4.3.6 Notation du type binaire codé décimal

6.4.3.6.1 Définition

Un entier non signé de 4 bits exprimant un chiffre décimal compris entre 0 et 9.

NOTE Ce type n’existe pas dans ASN.1.

6.4.3.6.2 Syntaxe

```
BinaryCodedDecimalType::= BCD4
```

6.4.3.6.3 Codage

Une valeur de BCD4 doit être codée par un entier non signé occupant la même place.

3	2	1	0
2^{3}	2^{2}	2^{1}	2^{0}

Plage: 0..9 (les autres valeurs ne sont pas définies)

EXEMPLE ‘0111’B = 7.

NOTE Certaines valeurs non définies peuvent être utilisées, par exemple, pour désigner le signe ou un autre opérateur arithmétique.

6.4.3.7 Notation des types unipolaires

6.4.3.7.1 Définition

Types primitifs ayant des valeurs distinctes qui ne sont pas négatives, interprétées comme un nombre entier divisé par une puissance fixe de deux, exprimant une valeur en pourcentage d’une plage.

NOTE Ces types n’existent pas en ASN.1, mais la CEI 870 les appelle ‘unsigned fixed point number’ (numéro de point fixe non signé).

6.4.3.7.2 Syntaxe

```
UnipolarType::= UNIPOLAR2.16
```

NOTE 1 Le chiffre devant la virgule indique le nombre de bits utilisés pour la puissance de deux qui forme la partie entière.

NOTE 2 Le facteur epsilon est égal à la valeur de la plus petite puissance de deux dans le nom (double octet).

6.4.3.7.3 Codage

Une variable de type unipolaire doit être transmise comme un entier non signé.

6.4.3.7.3.1 Codage de UNIPOLAR2.16

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

Plage: 0 .. 400 % – epsilon

6.4.3.8 Notation des types bipolaires

6.4.3.8.1 Définition

Types primitifs ayant des valeurs distinctes, positives ou négatives, traitées comme des nombres entiers (zéro inclus) divisés par une puissance fixe de deux, exprimant une valeur en pourcentage d’une plage.

NOTE Ces types n’existent pas en ASN.1, mais la CEI 60870-5-1 les appelle ‘signed fixed point number’ (numéro de point fixe signé).

6.4.3.8.2 Syntaxe

```
BipolarType::= BIPOLAR2.16 | BIPOLAR4.16
```

NOTE 1 Le numéro devant la virgule indique le nombre de puissances de 2 qui forment la partie entière.

NOTE 2 Le facteur epsilon est égal à la valeur de la plus petite puissance de deux dans le nom (double octet).

6.4.3.8.3 Codage

6.4.3.8.3.1 Codage de BIPOLAR2.16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
signe	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}
partie entière		partie fractionnelle													

Plage: -200 %..+200 % – epsilon

6.4.3.8.3.2 Codage de BIPOLAR4.16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
signe	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}
partie entière				partie fractionnelle											

Plage: -800 %..+800 % – epsilon

6.4.3.9 Notation du type réel

6.4.3.9.1 Définition

Type primitif dont les valeurs distinctes font partie d'un ensemble de nombres réels.

6.4.3.9.2 Syntaxe

```
RealType::= REAL32
```

6.4.3.9.3 Codage

Ce type doit être codé comme le prescrit l'IEEE 754 pour le format Short Real Number (32-bit).

NOTE 1 Il s'agit d'un type 'RealType' d'ASN.1, restreint au format IEEE 754 Short Real Number.

NOTE 2 Le type IEEE 754 de 64 bits en virgule flottante (REAL64) n'est pas considéré utile dans ce contexte.

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

Plage: $\pm$ 3,37 $\times$ 10^{+38}

6.4.3.10 Notation du type de caractère

6.4.3.10.1 Définition

Type primitif dont les valeurs distinctes sont membres du jeu de caractères défini dans l'ISO/CEI 8859-1.

6.4.3.10.2 Syntaxe

```
CharacterType::= CHARACTER8
```

6.4.3.10.3 Codage

Les caractères doivent être transmis dans un octet, sans bit de parité.

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

EXEMPLE ‘01100001’B = caractère ‘a’ selon l'ISO/CEI 8859-1.

6.4.3.11 Notation du type de caractère Unicode

6.4.3.11.1 Définition

Type primitif dont les valeurs distinctes sont membres du jeu de caractères défini dans l'ISO/CEI 10646.

6.4.3.11.2 Syntaxe

```
UnicodeType::= UNICODE16
```

6.4.3.11.3 Codage

Les caractères doivent être transmis dans deux octets.

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

6.4.3.12 Notation des types indéfinis

6.4.3.12.1 Définition

Type indéfini de contenu indéfini, mais ayant une taille fixe.

6.4.3.12.2 Syntaxe

```
AnyType::= WORD#, (# = entier positif non signé)
```

6.4.3.12.3 Codage

Une variable de type indéfini n’a pas de codage prescrit.

Les bits doivent être nommés en puissance de deux d'une variable de type UNSIGNED# qui occuperait cette place.

NOTE Ce nommage est dans la direction inverse de celle du décalage dans ce même mot.

6.4.3.12.3.1 Codage de WORD8

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

6.4.3.12.3.2 Codage de WORD16

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

6.4.4 Types structurés

6.4.4.1 Généralités

Cinq types structurés différents sont définis:

a) RECORD (taille variable),
b) ARRAY (taille fixe ou variable),
c) BITSET# (taille fixe),
d) ONE_OF (taille variable),
e) SOME_OF (taille variable).

6.4.4.2 Notation des types d'enregistrement

6.4.4.2.1 Définition

Type structuré défini en indiquant une liste fixe et ordonnée de types. Chaque valeur du nouveau type est une liste ordonnée de valeurs, une pour chacun des types de composant.

NOTE 1 Ce type est le 'Sequence Type' d'ASN.1, sans type optionnel.

NOTE 2 Lors de la définition d'un RECORD, il est recommandé de respecter l'alignement, c'est-à-dire qu'il convient de placer tous les éléments selon un décalage par rapport au début de l'enregistrement qui est un multiple de leur taille.

6.4.4.2.2 Syntaxe

```
RecordType::= RECORD { ElementTypeList }
```

avec

```
ElementTypeList::= ElementType | ElementTypeList, ElementType
```

et

```
ElementType::= identifier Type | Type
```

Les éléments d'un RECORD doivent être identifiés par l'indicatif du champ RECORD, suivi d'un point et de l'indicatif du sous-champ, qui peut être lui-même un type structuré.

EXEMPLE:

dossier.date.jour

6.4.4.2.3 Codage

Les éléments d'un RECORD doivent être transmis dans l'ordre de leur déclaration.

EXEMPLE Une valeur de type Date32 est représentée comme suit:

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

```
Date32::= RECORD
  {
  année        INTEGER16,
  factice  WORD4,
  mois     UNSIGNED4,
  jour       UNSIGNED8
  }
```

Le champ ‘factice’ a été introduit pour aligner le champ ‘jour’ sur une limite de mot de seize bits.

6.4.4.3 Notation pour les types bitset

6.4.4.3.1 Définition

Un ARRAY [#] de BOOLEAN1, ayant une taille fixe en bits définie par le suffixe #.

NOTE Ce type correspond au BITSTRING d’ASN.1.

6.4.4.3.2 Syntaxe

```
BitsetType::= BITSET# {NamedBitList}
```

avec

```
NamedBitList::= NamedBit | NamedBitList, NamedBit
```

et

```
NamedBit::= identifier (number) | identifier (DefinedValue)
```

a) La valeur de chaque ‘number’ ou ‘DefinedValue’ apparaissant dans la ‘NamedBitList’ doit être différente, et doit être le décalage d’un bit distinct dans la valeur du bitset.

b) Chaque ‘identifier’ apparaissant dans la ‘NamedBitList’ doit être différent.

c) Tous les éléments sont implicitement du type BOOLEAN1. La DefinedValue doit être TRUE (1) ou FALSE (0).

d) Les éléments doivent être déclarés en ordre de décalage croissant.

e) Si tous les éléments du BITSET sont déclarés, ‘number’ peut être omis. Il convient que cela soit le cas normal.

6.4.4.3.3 Codage

Les éléments d’un bitset doivent être transmis dans l’ordre de leur déclaration.

6.4.4.3.3.1 Codage de BITSET8

7	6	5	4	3	2	1	0
8e							1er

Plage: jeu de 8 bits du type booléen

EXEMPLE

```
AccessType8::= BITSET8
  {
  système      (0),                    -- 1er bit du bitset (bit de
                                          poids faible)
  propriétaire (1),
  groupe       (2),
  monde        (3),
  }
```

est équivalent à:

```
AccessType8::= BITSET8
  {
  système,              -- 1er bit du bitset (bit de poids faible)
  propriétaire,
  groupe,
  monde
  reserved4
  reserved5
  reserved6
  reserved7             -- 8e ou dernier bit du bitset (bit de poids
                                              fort)
  }
```

Un UNSIGNED8 occupant cette place et dont la valeur est ‘01’H signifie que ‘système’ est le seul membre du BITSET.

6.4.4.3.3.2 Codage de BITSET16

8ème							1er	16ème							9ème

EXEMPLE

```
AccessType::= BITSET16 {système (0), propriétaire (1), groupe (2),
                                              monde (3)}
```

La valeur ‘0000 0000 0000 0110’B signifie que ‘propriétaire’ et ‘groupe’ sont membres du bitset.

6.4.4.3.3.3 Codage de BITSET32

8e							1er	16ème							9ème
24ème							17ème	32ème							25ème

6.4.4.3.3.4 Codage de BITSET64

8e							1er	16ème							9ème
24ème							17ème	32ème							25ème
40ème							33ème	48ème							41ème
56ème							49ème	64ème							57ème

6.4.4.4 Notation du type tableau

6.4.4.4.1 Définition

Type structuré défini en référençant un seul type existant. Chaque valeur du nouveau type est une liste ordonnée de zéro, une ou plusieurs valeurs du type existant. La position de chaque valeur est identifiée par un index. Le nombre de valeurs est indiqué par une constante ou par un champ de la structure enveloppante. Ce nombre de valeurs peut être omis si une valeur de stoppage est définie.

NOTE Un ARRAY est un ‘SequenceOf Type’ d’ASN.1 comportant un certain nombre d’éléments indiqués par une constante, une variable dédiée ou aucun élément (élément de stoppage).

6.4.4.4.2 Syntaxe

```
ArrayType::= ARRAY [IndexList] OF Type

IndexList::= Index | IndexList, Index
Index::=
                      number | DefinedValue |
                      identifier |
                      identifier UnsignedType |
                      UnsignedType |
                      STOP = Value
```

La valeur ‘number’, ‘DefinedValue’ ou ‘identifier’ spécifient la taille du tableau en nombre d’éléments (0 pour un tableau vide). Son type doit être un entier non signé.

Si un type non signé est indiqué avec un ‘identifier’ défini, cela permet de déclarer le champ correspondant.

Si ‘identifier’ nomme un champ déclaré à l’extérieur du tableau, ledit champ doit se trouver dans la structure de données enveloppante au même niveau d’imbrication. Il peut également s’agir d’un sous-champ situé au même niveau d’imbrication, auquel cas le chemin complet doit être indiqué.

Si une valeur de stoppage est définie pour fermer un tableau ouvert, elle doit être du même type que l’élément du tableau.

La taille peut être indiquée par une expression arithmétique.

6.4.4.4.3 Codage

Les tableaux doivent être transmis en ordre croissant d’indice.

Les tableaux multidimensionnels doivent être transmis dans l’ordre dans lequel leurs index sont déclarés.

NOTE ARRAY OF [ligne, colonne] est transmis ligne par ligne.

Un tableau d’octets (de contenu indéfini, tel qu’un contenu de mémoire, par exemple) doit être transmis en ordre croissant d’adresse (ou d’index) de la mémoire d’application.

Tous les éléments du tableau doivent être transmis, même si certaines valeurs ne sont pas significatives.

EXEMPLE 1 Transmission d’un contenu de mémoire en octet:

```
DumpOctetType::= ARRAY [octet_count UNSIGNED16] OF WORD8.
```

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="16">octet_count</td></tr>
<tr><td colspan="8">octet à l’adresse M</td><td colspan="8">octet à l’adresse M+1</td></tr>
<tr><td colspan="16">octets</td></tr>
<tr><td colspan="8">octet à l’adresse (M + octet_count - 2)</td><td colspan="8">octet à l’adresse (M + octet_count - 1)</td></tr>
</table>

EXEMPLE 2 Transmission du même contenu de mémoire par mots de 16 bits, la valeur de word_count étant la moitié de celle d’octet_count de l’exemple précédent.

```
DumpWordType::= ARRAY [word_count UNSIGNED16] OF WORD16.
```

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="16">word_count</td></tr>
<tr><td colspan="16">mot à l'adresse M</td></tr>
<tr><td colspan="16">mot à l'adresse (M + word_count × 2 – 2)</td></tr>
</table>

EXEMPLE 3 Nombre d'éléments indiqué par un champ situé dans la structure de données enveloppante (décalage non spécifié):

```
DumpOctetType::= ARRAY [array_count] OF WORD8.
```

EXEMPLE 4 Nombre d'éléments indiqué par un champ d'une valeur structurée située dans la structure enveloppante:

```
HeaderType::= RECORD
  {
  name                  ARRAY [32] OF CHARACTER8
  bodysize              UNSIGNED16,
  ...
  }

FrameType::= RECORD
  {
  header                HeaderType
  body                  ARRAY [ header.bodysize ] OF CHARACTER8
  }
```

EXEMPLE 5 Chaînes de caractères, dans laquelle le caractère de stoppage est 'espace':

```
ProfibusString::= ARRAY [STOP = '20'H] OF CHARACTER8.
```

6.4.4.5 Notation des types de choix

6.4.4.5.1 Définition

Type structuré, défini en référençant une liste fixe, non ordonnée de types distincts. Chaque valeur du nouveau type est une valeur de l'un (exactement) des types de composants.

NOTE Ce type correspond au 'ChoiceType' d'ASN.1, mais il dispose d'une étiquette dédiée.

6.4.4.5.2 Syntaxe

```
OneOfType::= ONE_OF [identifier | identifier EnumeratedType]
                                            {AlternativeTypeList}
```

avec

```
AlternativeTypeList::= ElementType | ElementTypeList, ElementType
```

et

```
ElementType::= identifier [tag] Type | [tag] Type
```

et

```
tag::= UnsignedNumber | DefinedValue | identifier
```

Si une variable nommée est utilisée comme étiquette, elle doit appartenir à la structure enveloppante de l'élément.

Si la variable étiquetée est située au même niveau d'imbrication que le choix, seul le nom de la variable doit être inclus.

Si la variable est située à un niveau d'imbrication différent, le chemin permettant d'y accéder doit être inclus.

EXEMPLE 1 L'étiquette est un nombre (non recommandé car ce nombre doit être défini ailleurs):

```
Commands::= ONE_OF [choice_var ENUM8]
  {
  [3]                 OpenSequence,
  [2]                 CloseSequence,
  [5]                 StandbySequence
  }
```

EXEMPLE 2 L'étiquette est un type d'énumération situé dans les 16 bits précédant le choix:

```
CommandType::= ENUM16
  {
  OPEN                (3),
  CLOSE               (2),
  STANDBY             (5)
  }

Commands::= ONE_OF [choice_var CommandType]
  {
  [OPEN]              OpenSequence,
  [CLOSE]             CloseSequence,
  [STANDBY]           StandbySequence
  }
```

EXEMPLE 3 L'étiquette est définie au même niveau d'imbrication que la structure enveloppante:

```
Commands::= ONE_OF [choice_var]
  {
  [OPEN]              OpenSequence,
  [CLOSE]             CloseSequence,
  [STANDBY]           StandbySequence
  }

Command_Frame::== RECORD
  {
  choice_var          CommandType,
  ...
  command             Commands;
  ...
  }
```

EXEMPLE 4 L'étiquette est définie dans un sous-champ situé au même niveau d'imbrication:

```
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 Les chemins relatifs (par exemple -/-/header) ne sont pas recommandés.

6.4.4.5.3 Codage

Une valeur ONE_OF doit être codée en transmettant, avant la valeur, le champ de l'étiquette indiquant le choix réalisé.

La taille de la valeur transmise est implicite ou indiquée dans le type lui-même

NOTE ONE_OF est un SOME_OF comportant un seul élément.

EXEMPLE Une valeur particulière des choix de commandes ci-dessus est transmise de la manière suivante:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
choice_var (= 5)															
premier octet de (StandbySequence)															
…															
								dernier octet de (StandbySequence) ou factice							

6.4.4.6 Notation des types de jeu

6.4.4.6.1 Définition

Type structuré, défini en référençant une liste fixe non ordonnée de types distincts, dont certains peuvent être facultatifs. Chaque valeur du nouveau type est une liste non ordonnée de valeurs, une pour chacun des types de composants transmis.

NOTE Ce type correspond au 'SetType' d'ASN.1, mais avec une étiquette explicite.

6.4.4.6.2 Syntaxe

```
SetType::= SOME_OF { ElementTypeList}
```

avec

```
ElementTypeList::= ElementType | ElementTypeList, ElementType
```

et

```
ElementType::= [tag] NamedType
```

et

```
tag::= identifier | identifier ElementType | ElementType
```

Si l'étiquette est une variable nommée, la variable correspondante doit appartenir à une structure de données au même niveau d'imbrication ou à un sous-champ situé au même niveau d'imbrication, auquel cas elle doit être identifiée par son chemin complet.

Si le nombre de membres du jeu est fixe, le nom de référence peut être omis pour chaque membre dont la fonction est directement déductible de son type.

Si le sélecteur est un type énuméré, les constantes d'énumération utilisées pour sélectionner les éléments du jeu doivent être placées entre crochets.

Si une variable de type bitset fait office de sélecteur, elle doit être préalablement définie à l'intérieur de la structure de données enveloppante ou appartenir à un sous-champ d'un champ situé au même niveau d'imbrication, auquel cas son chemin complet doit être indiqué.

EXEMPLE 1 Étiquette de type entier non signé dans le champ précédant la valeur de l'ensemble:

```
MemberType::= SOME_OF [UNSIGNED8]
  {
  OPENSEQ              [3]                 Type_OpenSequence,
  CLOSESEQ             [2]                 Type_CloseSequence,
  STANDBY              [5]                 Type_StandbySequence
  }
```

EXEMPLE 2 Omission du nom de référence:

```
MemberType::= SOME_OF [UNSIGNED8]
  {
  [3]                  Type_OpenSequence,
  [2]                  Type_CloseSequence,
  [5]                  Type_StandbySequence
  }
```

EXEMPLE 3 Type énuméré comme étiquette (recommandé):

```
MemberType             ENUM8
  {
  OPENSEQ              (3),
  CLOSESEQ             (2),
  STANDBY              (5)
  }
..
CommandsType::= SOME_OF [MemberType]
  {
  [OPENSEQ]    Type_OpenSequence,
  [CLOSESEQ]   Type_CloseSequence,
  [STANDBY]    Type_StandbySequence
 }
```

EXEMPLE 4 Utilisation d'un bitset comme étiquette:

```
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
  ...
  }
```

6.4.4.6.3 Codage

Un jeu doit être codé en transmettant chaque valeur choisie.

Si une étiquette est indiquée, elle doit précéder chaque valeur sélectionnée, la valeur d'étiquette particulière 'FF'H (que des 1) fermant le jeu transmis.

Si l'étiquette est remplacée par un bitset, ce dernier doit être transmis avant le jeu, et les différents membres du jeu doivent être transmis de façon contiguë.

EXEMPLE 1 Une valeur particulière du jeu MemberType ci-dessus est transmise comme:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
member = '02'H								closesequence							
.. closesequence ..															
closesequence (end)								member ='05'H							
.. standbysequence ..															
Standbysequence								member = 'FF'H							

EXEMPLE 2 Si l'étiquette est remplacée par un bitset, le codage se présente de la manière suivante:

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

6.4.5 Alignement

Dans un type, il peut être nécessaire d'ajouter des bits de remplissage pour aligner le champ suivant sur une limite divisible par 16 ou 32 bits (ou sur toute autre limite). Pour exprimer ce remplissage, le qualificatif ALIGN est utilisé après le type. Les bits de remplissage ne sont pas définis (ils prennent la valeur 0 par défaut).

EXEMPLE La chaîne ci-dessous définit un tableau de caractères aligné sur une limite de 32 bits, quelle que soit la valeur de 'count':

```
AlignedString::= ARRAY ALIGN 32 [count] OF CHARACTER8.
```

6.4.6 Notation des types spéciaux

Un certain nombre de types structurés sont munis d'un identificateur de type spécial.

6.4.6.1 Notation du type de chaîne

STRING# est un ARRAY [] OF CHARACTER8, l'élément de stoppage devant être le caractère '00'H, et la taille réelle de la chaîne étant déduite du nombre de caractères significatifs, bien que le nombre de caractères transmis puisse être plus élevé.

EXEMPLE Une chaîne de type STRING32 est représentée par un ARRAY [32 STOP='00'H] OF CHARACTER8.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1er caractère ou '00'H								2ème caractère ou '00'H							
caractères...															
dernier caractère ou '00'H								32ème caractère ou '00'H							

6.4.6.2 Notation du type TIMEDATE48

6.4.6.2.1 Définition

Type structuré exprimant le temps absolu en secondes depuis le 1er janvier 1970, 0 heure 00, temps coordonné universel (UTC) (format Unix et ANSI-C).

NOTE Ce type est utilisé pour la distribution de l'heure, l'estampillage et la synchronisation.

6.4.6.2.2 Syntaxe

```
TimeDate48::= RECORD
  {
  seconds           SIGNED32,         -- temps écoulé depuis le 1er
                                         janvier 1970 à 0h00
  ticks             UNSIGNED16        -- fraction de seconde (1 tick
                                         = 1/65536s)
  }
```

6.4.6.2.3 Codage

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
secondes (plus significatives)															
secondes (moins significatives)															
ticks = 1/65536 s															

Le temps peut être représenté avec une granularité pouvant atteindre 15,3 µs (= 1/65536 s).

La plage est de 68 ans.

La précision de la partie fractionnelle doit être d'au moins 10 bits.

Les bits de poids faible non utilisés doivent être mis à zéro.

NOTE La variable TimeDate48 sera bouclée le 19 janvier 2038 à 3:14:07 UTC. Il convient de tenir compte de ce bouclage dans l'essai du logiciel.

6.4.6.3 Notation du type TIME64

6.4.6.3.1 Définition

Type structuré exprimant le temps absolu en secondes depuis le 1er janvier 1900, 0 heure 00, temps coordonné universel (UTC). Ce temps n'est pas compensé par des secondes intercalaires.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
secondes (plus significatives)															
secondes (moins significatives)															
ticks = 1/65536 s															
chirp = 0,232 µs															

NOTE 1 Cette définition provient du document Internet RFC1305, qui définit le protocole de synchronisation pour un réseau d'horloges distribuées. Il est différent du temps Unix, qui est basé sur l'année 1970.

NOTE 2 Une variable Time64 sera bouclée en 2036. Il convient de tenir compte de ce bouclage dans l'essai du logiciel. Par conséquent, la définition du temps peut également exprimer le temps restant jusqu'en janvier 2036.

6.4.6.4 Notation du type ASN.1 boolean8

6.4.6.4.1 Définition

Type primitif ayant deux valeurs distinctes, TRUE et FALSE.

NOTE Il s'agit du type 'BooleanType' d'ASN.1.

6.4.6.4.2 Syntaxe

```
Boolean8Type::= BOOLEAN8
```

6.4.6.4.3 Codage

Une variable de type booléen 8 doit être codée sur 8 bits, la valeur '00000000'B étant interprétée comme FALSE et toute autre valeur comme TRUE.

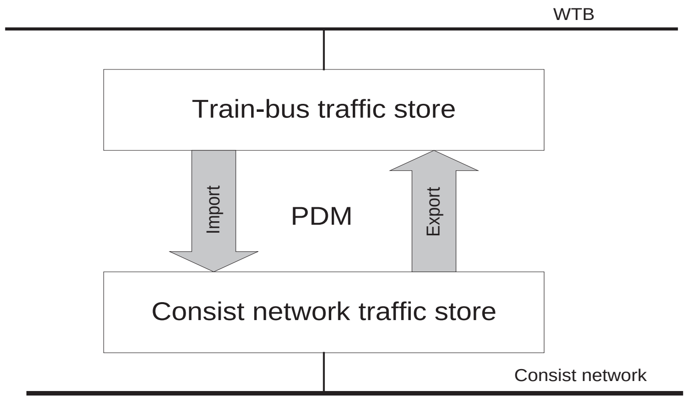
7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	FALSE
0	0	0	0	0	0	0	1	TRUE

7 Couche d'Application

7.1 Triage des Données de Processus

7.1.1 Types de triage

Le Triage de Données de Processus PDM (Process Data Marshalling) permet de copier des variables de processus d'un Traffic Store à l'autre:



Légende

Anglais	Français
Train bus traffic store	traffic_store du bus de train
consist network traffic store	traffic_store du réseau de rame
import	importation
export	exportation
consist network	réseau de rame

Figure 142 – Triage des Données de Processus

Deux types de triage sont définis:

– Triage d'exportation
– Triage d'importation

7.1.1.1 Triage d'exportation

Le triage d'exportation permet de copier les variables d'un ou de plusieurs Traffic Stores de Réseaux de Rame vers le port émetteur du Traffic Store WTB. La totalité du port WTB est écrite. L'espace inutilisé du port doit donc être rempli par des valeurs par défaut. Le triage d'exportation peut traiter certaines variables de processus.

Le triage d'exportation détermine la longueur de la trame exportée en fonction du type de trame.

7.1.1.2 Triage d'importation

Le triage d'importation permet de copier les variables du Traffic Store WTB vers le (les) Traffic Store(s) configuré(s) du point de vue statistique du Réseau de Rame. Le triage d'importation peut traiter certaines variables de processus.

7.1.2 Modes de triage

Une rame peut comporter différents modes d'exploitation dynamiques. Selon ces modes, PDM propose des modes de triage. Chaque mode de triage peut faire l'objet d'une configuration différente.

EXEMPLE Une rame de traction (une locomotive, par exemple) peut être l'élément principal de traction d'une rame, d'un élément de traction suiveur ou ne pas assurer la traction du tout. Dans chaque cas, différentes données sont importées et exportées.

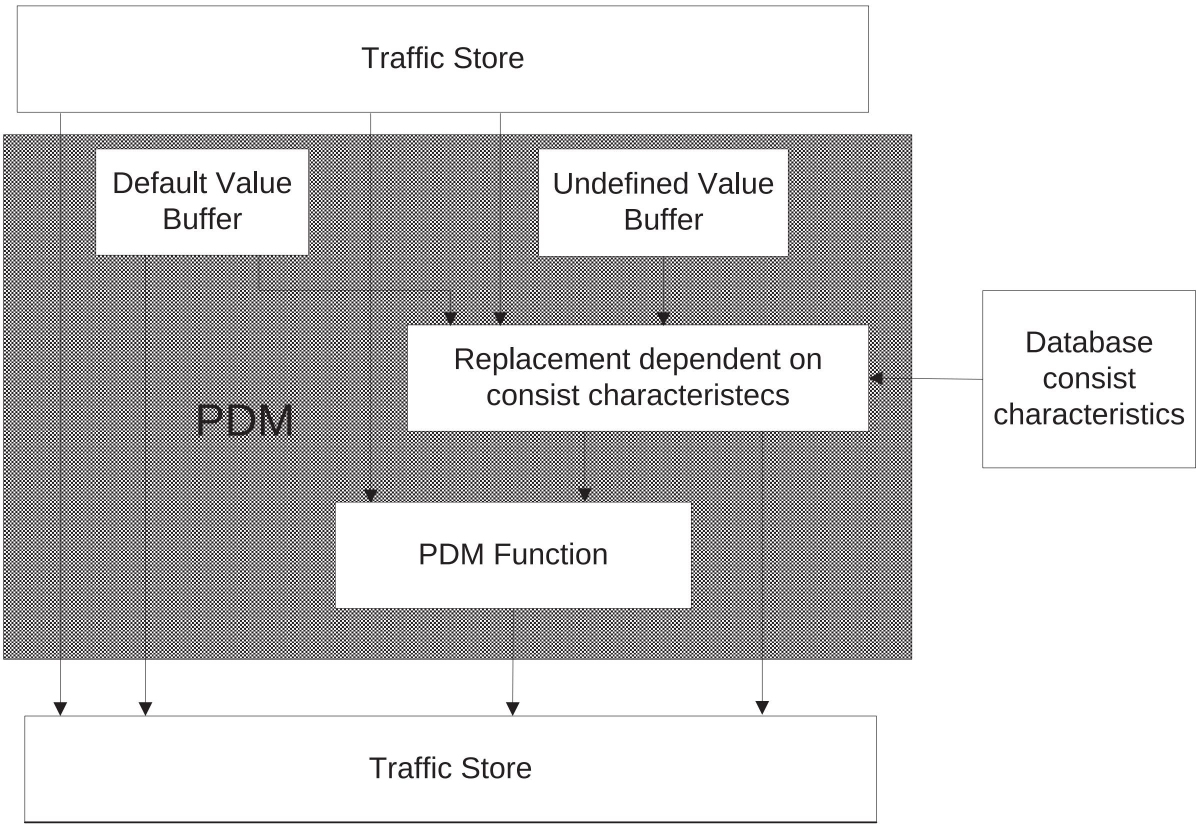
Il convient de toujours prévoir un mode par défaut si aucun mode particulier n'est utilisé pour PDM.

Si le mode d'exploitation de la rame change, PDM peut être configuré pour accepter le nouveau mode de triage.

7.1.3 Chemins d'accès aux données dans PDM

La présente section est donnée à titre d'information uniquement et n'est pas normative, chaque fournisseur de passerelle pouvant la mettre en œuvre de manière différente.

PDM trie les données de processus à partir d'une source vers une destination. La destination est toujours un Traffic Store. La source est un Traffic Store, un tampon de valeur par défaut ou un tampon de valeur indéfinie.



Légende

Anglais	Français
Traffic Store	Traffic Store
Default Value Buffer	Tampon de valeur par défaut
Undefined Value Buffer	Tampon de valeur indéfinie
Replacement dependent on consist characteristics	Remplacement dépendant des caractéristiques de la rame
Database consist characteristics	Base de données Caractéristiques de la rame
PDM Function	Fonction PDM

Figure 143 – Chemins d'accès PDM

Les chemins d'accès aux données suivants peuvent être configurés:

entre Traffic Stores, directement ou par l'intermédiaire d'une fonction PDM. Il s'agit d'une tâche de base de PDM. La variable de processus peut être traitée par une fonction avant d'être écrite dans le Traffic Store de destination.

d'un tampon de valeur par défaut à un Traffic Store, les valeurs par défaut sont utilisées s'il convient d'écrire une valeur dans un port, mais il n'y a en aucun cas une variable de processus qui peut fournir cette valeur. Les valeurs par défaut n'ont pas pour objet de remplacer les variables de processus non valides ou trop anciennes.

variable de processus dépendant des caractéristiques de rame entre Traffic Stores, directement ou par l'intermédiaire d'une fonction PDM. Les variables de processus peuvent être combinées à des caractéristiques de rame statiques et dynamiques. Le triage n'a lieu qu'en présence des caractéristiques de rame. Sinon, le PDM remplace la variable de processus par une valeur par défaut indéfinie ou définie par l'application du type de données appropriée. Une valeur indéfinie est représentée par une variable de contrôle à laquelle a été attribuée la valeur 11_b ou par la valeur de variable définie entièrement par des "1",. (conformément à la norme TCN CEI 61375).

Il convient de considérer trois cas pour les variables de processus dépendantes de la rame:

Une variable dotée d'une variable de contrôle: PDM peut remplacer une valeur indéfinie en attribuant la valeur 11_b à la valeur de contrôle.

Plusieurs variables dotées d'une variable de contrôle: si toutes les variables ne sont pas prises en charge, PDM peut remplacer les valeurs indéfinies en attribuant la valeur 11_b à la valeur de contrôle.

Plusieurs variables dotées d'une variable de contrôle: si seules quelques variables ne sont pas prises en charge, il convient que PDM remplace les valeurs par défaut définies par l'application.

La variable de processus peut être traitée par une fonction avant d'être écrite dans le Traffic Store de destination. Si la variable de processus est remplacée par une valeur indéfinie, elle peut être ignorée comme argument de fonction.

7.1.4 Fonctionnement de PDM

PDM est activé par un temporisateur configurable ou un événement (réception de données WTB, par exemple). Après l'activation, PDM copie toutes les variables configurées pour le type de triage activé.

Le processus de copie comporte quatre étapes:

a) Toutes les variables sont lues, les données étant définies l'une après l'autre.
b) Pour chaque variable dont la supervision de rafraîchissement est configurée, le rafraîchissement est vérifié. Les variables trop anciennes sont invalides (voir ci-dessous).
c) Toutes les fonctions configurées (voir 7.1.5 Fonctions du PDM) sont donc appliquées aux variables.
d) Enfin PDM copie les variables, les résultats de la fonction et les valeurs par défaut dans les ports de destination et les Traffic Stores.

- *pour tous* datasets d'origine des Variables
  - lire toutes les variables du dataset
  - *si* (Triage d'importation *et* Champ de type de trame ok) *ou non* Triage d'importation
    - *alors*
      - *pour toutes* variables du dataset
        - *si* vérifier le rafraîchissement, données trop anciennes
          - *alors*
            - déclarer la variable non valide
          - *sinon*
    - *sinon*
      - définir toutes les variables du dataset non valides
- *si* Triage d'importation *et* tout Champ de type de trame pas ok
  - *alors*
    - déclarer toutes les variables du Traffic Store non valide du WTB
  - *sinon*
- *pour toutes* Fonctions PDM
  - *executer* Fonction PDM
- *pour tous* datasets de destination des Variables
  - écrire toutes les variables du dataset
  - (Variables PD, Résultats de la fonction, Valeurs par défaut)

Figure 144 – Fonctionnement de PDM

Une variable ou le résultat d'une fonction (voir 7.1.5) est invalidé par l'algorithme ci-dessous:

si variable ou résultat de la fonction comporte une variable de contrôle	
alors attribuer la valeur 00_b au contrôle	*sinon* attribuer la valeur "0" à la variable

Figure 145 – PDM invalide la variable ou le résultat de la fonction

Si la variable ou le résultat de la fonction comporte une variable de contrôle, la valeur 00_b est attribuée à la variable de contrôle uniquement. La valeur de la variable ne peut pas être définie entièrement par des '0', étant donné que la variable ne peut pas être trop invalide.

Si la variable ou le résultat de la fonction ne comporte pas de variable de contrôle, sa valeur est définie entièrement par des '0'. Ceci est conforme à la présente norme.

NOTE Dans la mesure où l'écrasement de la valeur d'une Process_Variable entièrement par des '0' ou entièrement par des '1' peut produire une valeur correcte, une Check_Variable du même Dataset sert d'indicateur de validité, en cas de problème (voir 6.2.2.2.3).

7.1.5 Fonctions du PDM

7.1.5.1 Généralités

Outre la pure copie des variables, PDM prend en charge le traitement des variables de processus par les fonctions. Le traitement des fonctions est pris en charge pour tous les types de triage.

EXEMPLE Soit une application souhaitant savoir si toutes les portes d'un train sont fermées. Etant donné que le train peut être composé de 1 à 20 rames équipées de portes, l'application doit être en mesure de traiter un large éventail de données d'entrée. L'utilisation d'une fonction du PDM permettant de lire tous les états des portes et de générer une variable signalant que "toutes les portes sont fermées" facilite la programmation de l'application.

En règle générale, les fonctions proposées par PDM se présentent sous la forme suivante:

$y = f(x_1, x_2, \ldots x_n)$; x_i , i = 1 .. n, argument d'entrée, y résultat de la fonction

Il peut exister un certain nombre d'arguments (supérieur à zéro) et un résultat. Les arguments d'une fonction peuvent provenir de différents ports et Traffic Stores. Les arguments et le résultat sont décrits par des PV_Names.

Le PDM offre les fonctions de traitement standard suivantes:

- fonctions logiques:
  AND, AND_IGNORE_INVALID
  OR, OR_IGNORE_INVALID
  XOR, XOR_IGNORE_INVALID

- fonctions numériques:
  MIN, MIN_IGNORE_INVALID
  MAX, MAX_IGNORE_INVALID
  SUM, SUM_IGNORE_INVALID

7.1.5.2 Traitement de la fonction

La validité de tous les arguments d'une fonction est vérifiée. Les variables trop anciennes sont déjà invalidées par la deuxième étape du processus de copie. Les arguments peuvent comporter ou non une variable de contrôle. Les deux variantes peuvent être mélangées si la fonction comporte plusieurs arguments.

Si un argument est invalide ou indéfini, il peut être ignoré (fonctions XXX_IGNORE_INVALID). Si un argument invalide ou indéfini n'est pas ignoré, le résultat de la fonction qu'il génère est non valide.

Seuls les arguments valides sont traités. Si tous les arguments sont invalides ou indéfinis, le résultat est invalide.

Le cas échéant, le type des arguments est converti en un type adapté au traitement. Le type de traitement peut être configuré.

Si une erreur se produit pendant l'évaluation de la fonction, le résultat est invalide.

A l'issue du traitement de tous les arguments, le résultat calculé est converti dans la fonction souhaitée décrite par un PV_Name.

Le résultat de la fonction peut comporter ou non une variable de contrôle, indépendante des arguments.

<table>
<tr><td colspan="4">pour tous arguments de la fonction</td></tr>
<tr><td rowspan="3"></td><td colspan="3">si variable est valide</td></tr>
<tr><td>alors
diffuser le type d'arguments vers le type de calcul</td><td colspan="2">sinon
si IGNORE_INVALID</td></tr>
<tr><td>appliquer la fonction à l'argument et calculer le résultat de la fonction</td><td>alors
ignorer l'argument</td><td>sinon
déclarer le résultat de la fonction invalide
renvoyer résultat de la fonction</td></tr>
<tr><td colspan="4">si toutes les variables sont invalides</td></tr>
<tr><td colspan="3">alors
déclarer le résultat de la fonction invalide</td><td rowspan="2">sinon</td></tr>
<tr><td colspan="3">renvoyer résultat de la fonction</td></tr>
<tr><td colspan="4">diffuser le type de calcul vers le type de résultat</td></tr>
<tr><td colspan="4">si résultat de la fonction comporte un contrôle</td></tr>
<tr><td colspan="3">alors
attribuer la valeur 01_b au contrôle</td><td>sinon</td></tr>
<tr><td colspan="4">renvoyer résultat de la fonction</td></tr>
</table>

Figure 146 – Fonctionnement de PDM

La validité d'une variable est contrôlée par l'algorithme suivant:

<table>
<tr><td colspan="4">si Variable comporte un contrôle</td></tr>
<tr><td colspan="2">alors</td><td colspan="2">sinon</td></tr>
<tr><td colspan="2">si contrôle = 10_b or contrôle = 01_b</td><td colspan="2">si valeur de Variable comporte "0" ou "1"</td></tr>
<tr><td>alors
La variable est valide</td><td>sinon
La variable est invalide</td><td>alors
La variable est invalide</td><td>sinon
La variable est valide</td></tr>
</table>

Figure 147 – Contrôle de validité du PDM

7.1.5.3 Fonctions logiques: AND, OR et XOR

Pour ces fonctions, les arguments sont de type BOOLEAN, ANTIVALENT ou BITSET. Si l'argument est de type BITSET, il est nécessaire de spécifier également la position du bit à l'intérieur du bitset. Les types d'arguments peuvent être mélangés à l'intérieur d'un appel de fonction.

La valeur du résultat de ces fonctions est de type BOOLEAN ou ANTIVALENT. De plus, il est possible de spécifier si la variable est utilisée directement ou si elle est ignorée avant utilisation.

7.1.5.4 Fonctions numériques: MIN, MAX, SUM

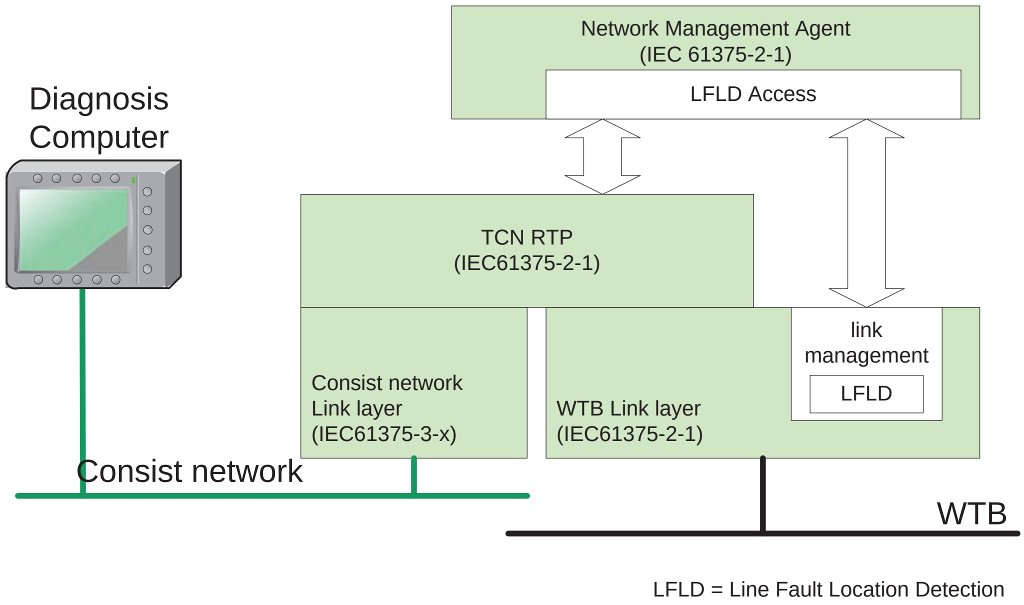
Pour ces fonctions, les arguments sont de type INTEGER, UNSIGNED, REAL et FRACTIONAL, avec une taille minimale de 32 bits. INTEGER et UNSIGNED de différentes tailles peuvent être mélangés à l'intérieur d'un appel de fonction. A l'issue du traitement, le type de résultat est converti (par diffusion) en type de variable de destination. Il n'y a pas de vérification de plage, faire attention aux débordements.

7.2 Détection de l'emplacement de défaut en ligne du WTB

Un système de diagnostic peut surveiller les bits de perturbation de ligne du WTB dans le mot d'état du nœud de passerelle. Si le système de diagnostic détecte une perturbation permanante sur une ligne, il peut demander une Détection de l'emplacement de défaut en ligne (LFLD) au niveau de la Gestion de Réseau TCN (TNM). La Gestion de Réseau TCN coopère avec la gestion de la couche de liaison du WTB afin de détecter l'emplacement du défaut. A l'issue du processus LFLD, le système de diagnostic peut lire le résultat au niveau de la TNM.

Un défaut qui gêne le bon déroulement de la ligne peut engendrer une réflexion du signal et gêner la communication de plusieurs nœuds sur cette ligne. Ainsi, le processus LFLD ouvre les relais d'interruption de la ligne perturbée au niveau des nœuds intermédiaires. Cela génère des segments correctement terminés du WTB, qui peuvent être contrôlés.

7.2.1 Architecture



Légende

Anglais	Français
Network Management Agent (IEC 61375-2-1)	Agent de gestion de réseau (CEI 61375-2-1)
LFLD Access	Accès LFLD
Diagnosis Computer	Système de diagnostic
TCN RTP (IEC 61375-2-1)	TCN RTP (CEI 61375-2-1)
Consist network Link layer (IEC 61375-3-x)	Couche de liaison du réseau de rame (CEI 61375-3-x)
WTB Link layer (IEC 61375-2-1)	Couche de liaison du WTB (CEI 61375-2-1)
link management	gestionnaire de liaison
LFLD = Line Fault Location Detection	LFLD = Détection de l'emplacement de défaut en ligne
Consist Network	Réseau de rame

Figure 148 – Architecture LFLD

La Détection de l'emplacement de défaut en ligne du WTB (LFLD) est une fonction de la couche de liaison du WTB, mise en œuvre dans la gestion de liaison. La gestion de liaison utilise la TNM pour communiquer avec les autres nœuds du WTB pendant le processus LFLD.

La Gestion de Réseau TCN offre un nouveau service LFLD dans les services de liaison du WTB (CEI 61375-1, 8.4.3.6) et prend en charge les sous-commandes pour le système de diagnostic et pour la gestion de liaison du WTB.

Le système de diagnostic peut accéder au service TNM LFLD par données de message TCN par l'intermédiaire du MVB pour:

- démarrer LFLD sur un nœud d'extrémité WTB
- Extraire le résultat LFLD
- Annuler LFLD

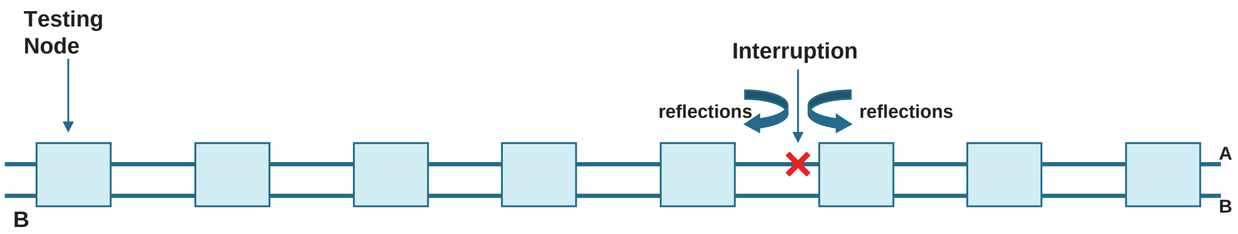
Le gestionnaire de liaison WTB accède au service TNM LFLD par données de message TCN par l'intermédiaire du MVB pour

- arrêter et démarrer le processus LFLD sur l'autre nœud d'extrémité WTB
- arrêter et démarrer le processus LFLD sur les nœuds intermédiaires WTB

7.2.2 Présentation du protocole

Etape 1:

Le système de diagnostic demande un LFLD sur un nœud d'extrémité, qui devient un nœud d'essai. Le gestionnaire de liaison WTB du nœud d'essai lance le processus de détection

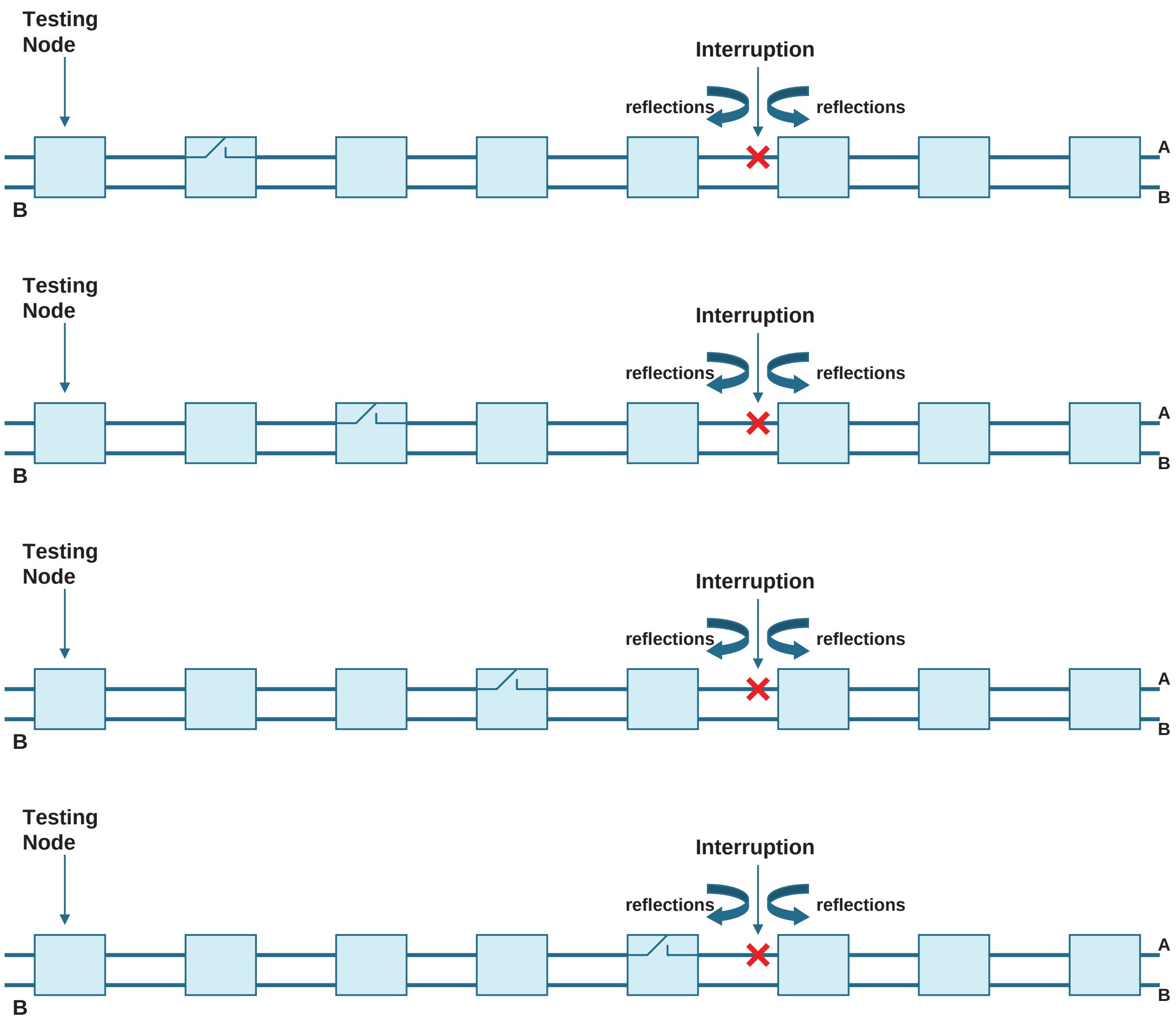


Légende

Anglais	Français
Testing Node	Nœud d'essai
reflections	réflexions

Etape 2:

Le nœud d'essai demande au nœud suivant (le nœud de segmentation) d'ouvrir le commutateur de ligne. Si le nœud d'essai n'enregistre aucun défaut de ligne pour les trames provenant du nœud de segmentation pendant une durée T_2, ce segment de ligne est supposé fonctionner correctement. Ensuite, le nœud d'essai demande au nœud de fermer de nouveau le commutateur de ligne et passe au seul nœud suivant. Ce processus se déroule tant que le nœud d'essai n'enregistre pas les défauts de ligne provenant du nœud de segmentation.

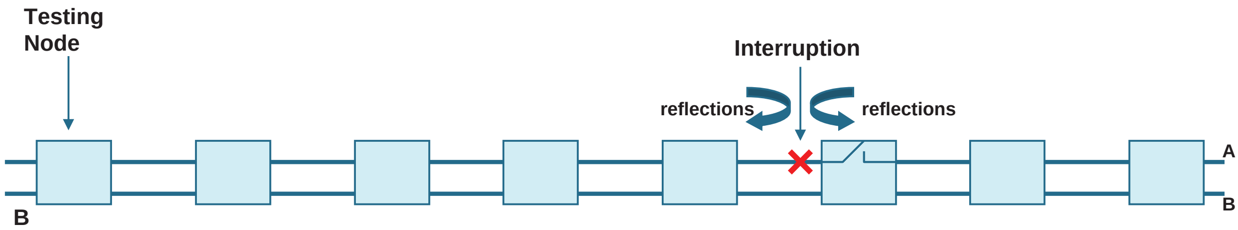


Légende

Anglais	Français
Testing Node	Nœud d'essai
reflections	réflexions

Phase 3:

A présent, le nœud d'essai ne reçoit plus sans défaut de ligne provenant du nœud de segmentation. Cela indique qu'iml doit y avoir un défaut sur le dernier segment soumis à essai.

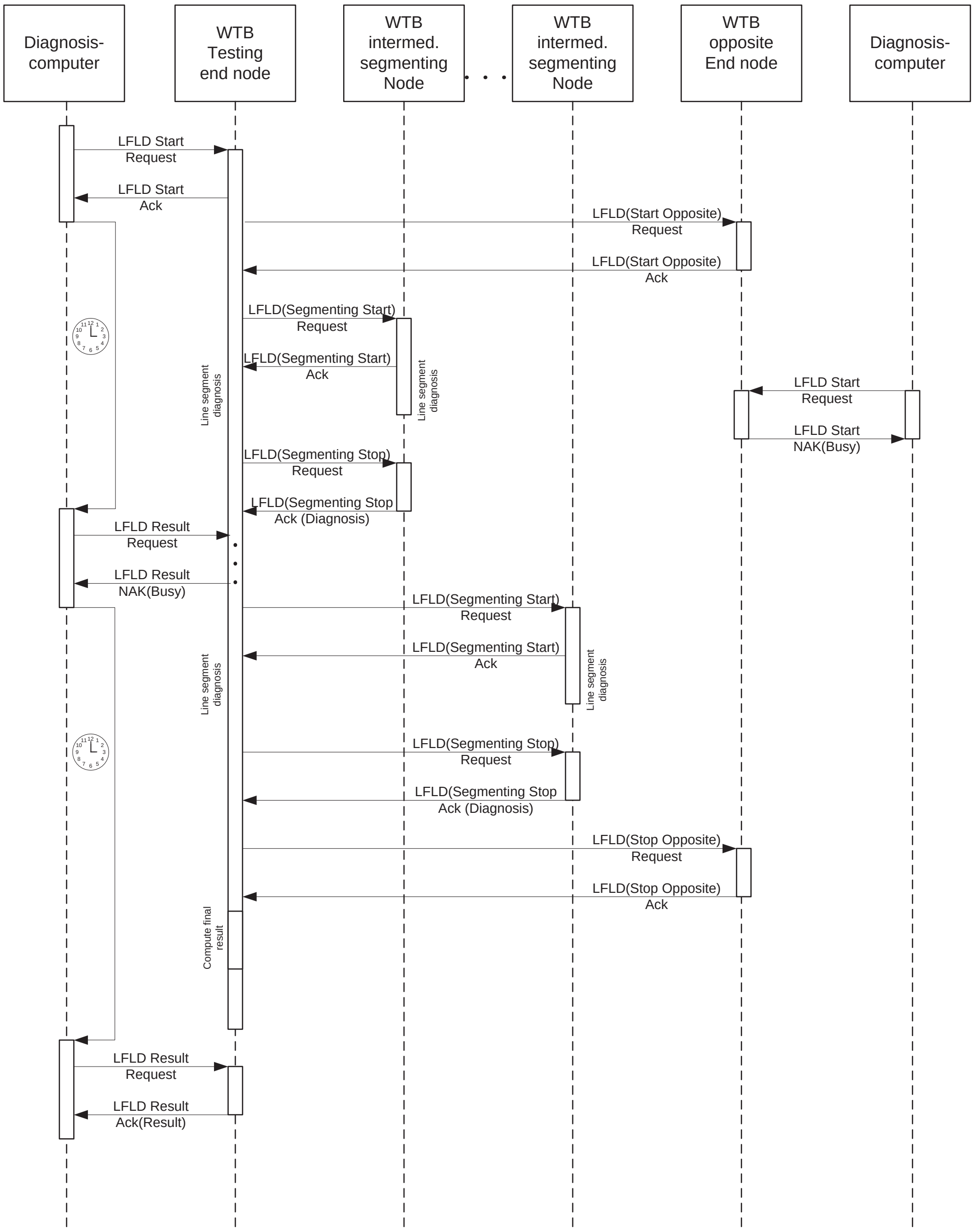


Légende

Anglais	Français
Testing Node	Nœud d'essai
reflections	réflexions

7.2.3 Séquence LFLD

La Figure 149 illustre la séquence de processus LFLD.



Légende

Anglais	Français
WTB intermed. segmenting Node	Nœud de segmentation intermed. du WTB
Diagnosis computer	système de diagnostic
WTB opposite End node	Nœud d'extrémité opposé du WTB
WTB Testing end node	Nœud d'extrémité d'essai du WTB
LFLD Start Request	Demande de démarrage LFLD
LFLD(Start Opposite)	LFLD (démarrage à l'opposé)
Request	Demande
LFLD(Start Opposite)	LFLD (démarrage à l'opposé)
Ack	Accusé de réception
LFLD Start Ack	LFLD (accusé de réception de démarrage)
LFLD (Segmenting Start)	LFLD (Démarrage de segmentation)
LFLD Start NAK(Busy)	LFLD NAK de démarrage (Occupé)
LFLD(Segmenting Stop)	LFLD (Arrêt de segmentation)
Ack (Diagnosis)	Accusé de réception (Diagnostic)
LFLD Result Request	Demande de résultat LFLD
LFLD Result	Résultat LFLD
NAK(Busy)	NAK (Occupé)
Line segment diagnosis	Diagnostic du segment de ligne
LFLD(Stop Opposite)	LFLD (Arrêt à l'opposé)
Computer final result	résultat final du système
LFLD Result	Résultat LFLD
Ack(Result)	Accusé de réception (Résultat)

Figure 149 – Séquence LFLD

Le système de diagnostic lance le processus LFLD par un message TNM.

Le nœud d'essai (TN) envoie un message TNM pour informer le nœud d'extrémité opposé que le processus LFLD est démarré. Cette opération doit empêcher le nœud d'extrémité opposé de démarrer également un processus LFLD.

Lorsque le nœud d'essai reçoit le message de réponse de la part du nœud d'extrémité opposé, il sélectionne le premier nœud de segmentation.

Le nœud d'essai demande au nœud de segmentation de segmenter le bus (relais d'interruption ouvert) et de démarrer le diagnostic de ligne.

Lorsque le nœud d'essai reçoit le message de réponse de la part du nœud de segmentation, il supervise le segment du nœud d'essai au nœud de segmentation, s'il peut recevoir une trame de réponse des Données de Processus provenant du nœud de segmentation ou du seul nœud précédent sans perturbation de ligne.

Le nœud de segmentation supervise le segment du nœud de segmentation au nœud d'extrémité opposé pour recevoir une trame de réponse de Données de Processus provenant du nœud d'extrémité opposé sans perturbation.

A l'issue de la période individuelle la plus longue possible (128 * 25 msec), le nœud d'essai demande au nœud de segmentation d'arrêter la segmentation du bus (relais d'interruption fermé) et d'indiquer s'il a pu recevoir du nœud d'extrémité opposé sans perturbation.

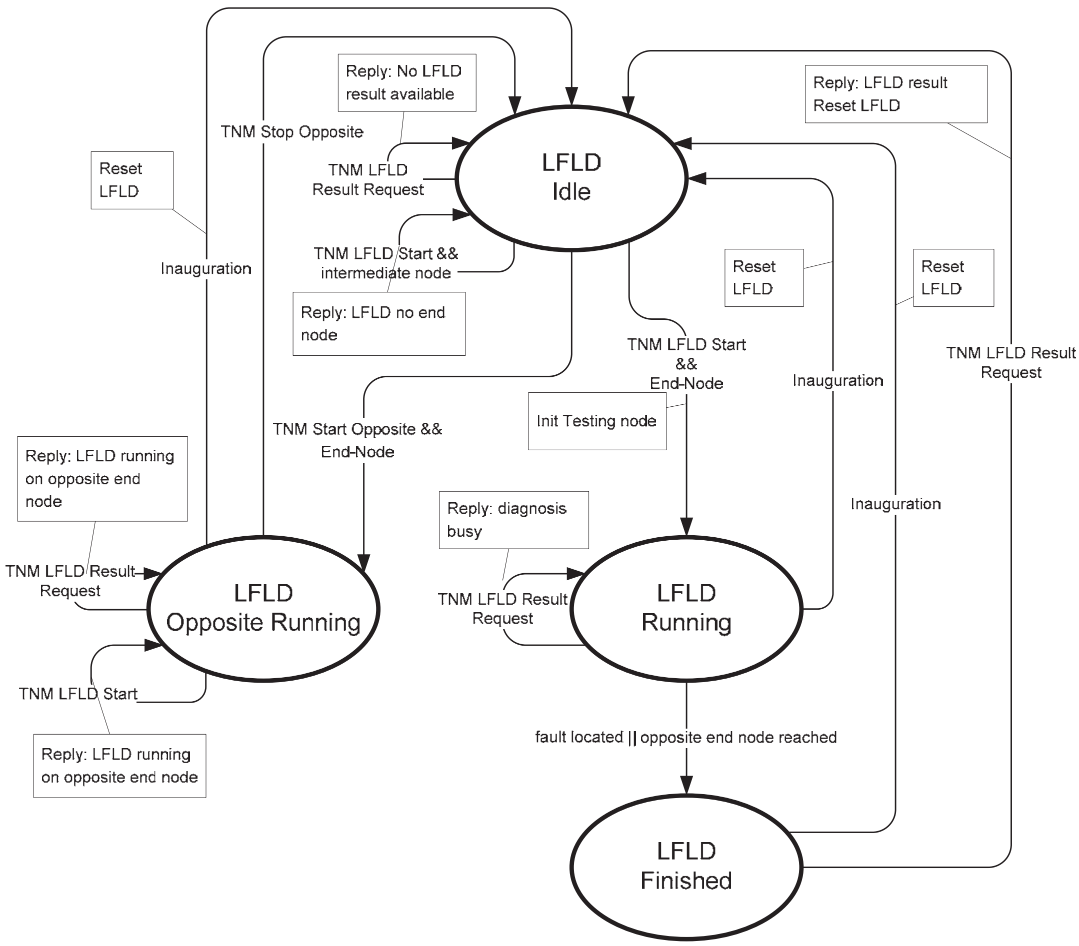
Le nœud d'essai sélectionne le nœud de segmentation suivant et répète les étapes 4) à 7) tant que l'emplacement ou le défaut n'a pas été détecté ou que le nœud d'extrémité opposé n'a pas été atteint.

Lorsque le nœud d'extrémité est atteint, le nœud d'essai l'informe que le processus LFLD est terminé.

Le système de diagnostic peut extraire le résultat LFLD du nœud d'essai. Si le processus LFLD est en cours, le nœud d'essai refuse la demande avec un message d'erreur. Le système de diagnostic doit interroger le nœud d'essai tant que le résultat n'a pas été obtenu.

Si un système de diagnostic demande un LFLD au niveau du nœud d'extrémité opposé lors du processus LFLD, le nœud d'extrémité opposé refuse la demande avec un message d'erreur.

7.2.4 Machine d'état du nœud d'extrémité (Nœud d'essai)



Légende

Anglais	Français
Reply: No LFLD result available	Réponse: aucun résultat LFLD disponible
Reply: LFLD result	Réponse: résultat LFLD
Reset LFLD	Réinitialiser LFLD
TNM Stop Opposite	Arrêt TNM à l'opposé
LFLD Idle	LFLD en veille
Inauguration	Inauguration
TNM LFLD Result Request	Demande de résultat LFLD du TNM
TNM LFLD Start && intermediate node	Début LFLD du TNM sur nœud intermédiaire
Reply: LFLD no end node	Réponse: LFLD pas sur nœud d'extrémité
TNM LFLD Start && End-node	Début LFLD du TNM sur nœud d'extrémité
Reply: LFLD running on opposite end node	Réponse: LFLD en cours d'exécution sur nœud d'extrémité opposé
Init Testing node	Nœud d'extrémité init.
LFLD Opposite Running	LFLD en cours d'exécution à l'opposé
Reply: diagnosis busy	Réponse: diagnostic occupé
LFLD Running	LFLD en cours d'exécution
TNM LFLD Start	Début LFLD du TNM
Reply: LFLD running on end node	Réponse: LFLD en cours d'exécution sur nœud d'extrémité
fault located	défaut détecté
opposite end node reached	nœud d'extrémité opposé atteint
LFLD Finished	LFLD terminé

Figure 150 – Machine d'état du nœud d'extrémité

7.2.5 Machine d'état du nœud intermédiaire (Nœud de segmentation)

La mise en œuvre du nœud de segmentation est sans état.

7.2.6 Sélection de ligne perturbée

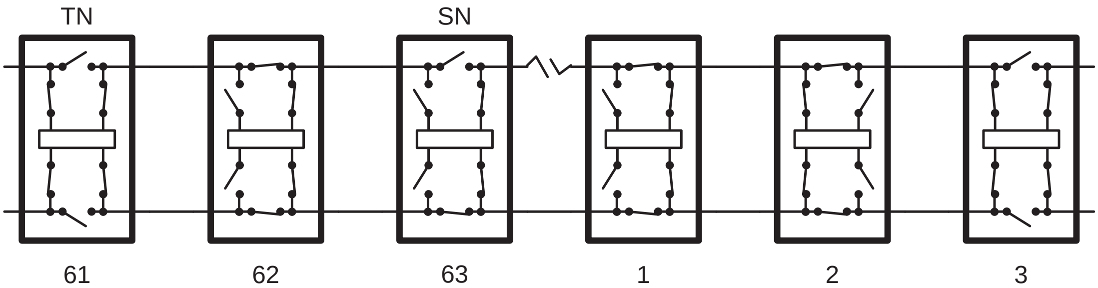
La ligne perturbée A ou B dépend de l'orientation de chaque nœud.

Le nœud d'essai a sa ligne perturbée locale. Le nœud d'essai déduit la ligne perturbée du maître WTB selon l'orientation propre par rapport au maître WTB et l'utilise comme ligne perturbée normalisée. Lorsque le nœud d'essai démarre et arrête un nœud de segmentation, il demande l'ouverture et la fermeture de la ligne perturbée normalisée. Le nœud de segmentation déduit de la ligne perturbée normalisée et de son orientation par rapport au maître, quel relais de ligne ouvrir et fermer.

7.2.7 Détection de l'emplacement

Le nœud d'essai initialise cette paire de nœuds avec le nœud d'essai et son voisin direct. Les étapes du processus LFLD se déroulent nœud par nœud.

EXEMPLE Le processus LFLD a atteint le nœud de segmentation 63 et le défaut est compris entre 63 et 1.



Légende

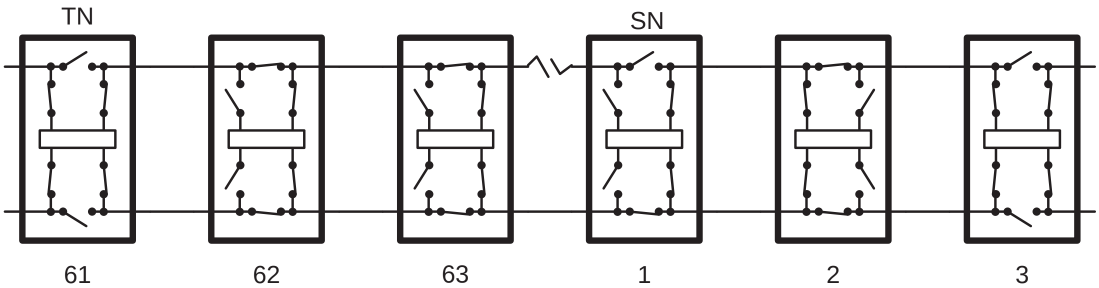
TN nœud d'essai
SN nœud de segmentation

Figure 151 – Processus LFLD, nœud de segmentation au nœud 63

Dans la Figure 151, le nœud d'essai ne peut pas atteindre le nœud de segmentation 63 car le relais de raccordement est fermé sur le côté éloigné. Le nœud d'essai peut uniquement voir une bonne réponse provenant du nœud précédent. Le nœud d'essai ne peut donc pas distinguer si ce défaut se situe entre les nœuds 62 et 63 ou entre les nœuds 63 et 1.

Le nœud de segmentation supervise donc le segment WTB de l'autre côté du relais d'interruption (entre les nœuds 63 et 3, dans cet exemple), puis indique la qualité du segment au nœud d'essai lorsque ce dernier demande de fermer le relais d'interruption. Dans cet exemple, le nœud 63 détecte la mauvaise qualité du segment, le nœud d'essai pouvant supposer la bonne qualité du segment des nœuds 61 à 63, même s'il ne peut obtenir une réponse du nœud 63.

Le nœud d'essai passe donc au nœud de segmentation suivant (nœud 1).



Légende

TN nœud d'essai
SN nœud de segmentation

Figure 152 – Processus LFLD, nœud de segmentation au nœud 1

Le nœud de segmentation (désormais nœud 1) supervise le segment du nœud 1 au nœud 3, le détecte comme étant bon et en informe le nœud d'essai. Le nœud d'essai peut déduire qu'un défaut s'est produit juste avant le nœud de segmentation, c'est-à-dire entre les nœuds 63 et 1.

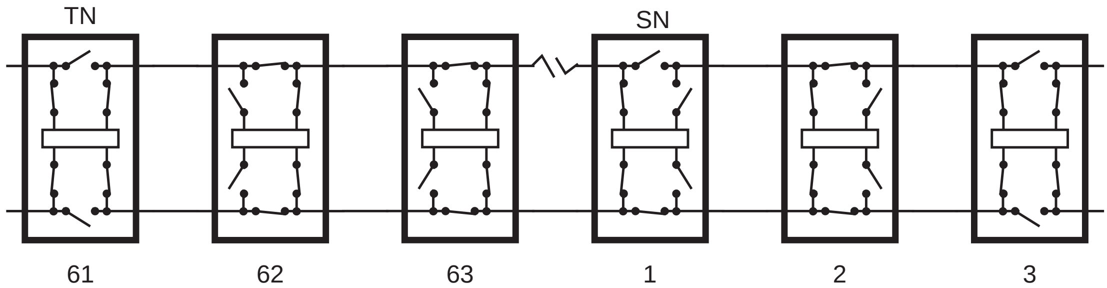
Lorsque le nœud d'essai obtient le rapport du nœud de segmentation, il décide:

si le nœud de segmentation signale qu'il a reçu un nœud d'extrémité opposé sans perturbation, d'arrêter le processus LFLD, et la paire de nœuds comprend le nœud de segmentation et le nœud qui le précède immédiatement.

si le nœud d'essai reçoit le nœud de segmentation ou le nœud qui le précède immédiatement sans perturbation, de définir la paire de nœuds comme le nœud reçu et le nœud suivant.

Cette décision est prise après chaque arrêt d'un nœud de segmentation.

Lorsque le nœud d'essai est arrêté ou s'il atteint le nœud d'extrémité opposé, il doit s'assurer qu'il a bien résolu le cas suivant:



Légende

TN nœud d'essai
SN nœud de segmentation

Figure 153 – Processus LFLD, nœud de segmentation au nœud 1, connexion dans la direction 1

Le nœud d'essai ne peut atteindre ni le nœud 63 ni le nœud 1, et le nœud de segmentation ne peut pas atteindre le nœud 3. Cependant, le nœud d'essai peut déduire que le défaut doit se trouver entre les nœuds 63 et 1, car si ce n'était pas le cas, il aurait détecté le segment compris entre le nœud 61 et 1 comme étant de bonne qualité.

Le nœud d'essai décide:

si la paire de nœuds est (63, 1) et si le nœud d'essai se trouve au niveau de l'adresse inférieure et le relais de raccordement du maître WTB dans la direction 2, de corriger la paire de nœuds en (1, 2)

si la paire de nœuds est (1, 2) et si le nœud d'essai se trouve au niveau de l'adresse supérieure et le relais de raccordement du maître WTB dans la direction 1, de corriger la paire de nœuds en (63, 1)

Le résultat final du processus LFLD est une paire de nœuds qui délimite l'emplacement du défaut. Il est enregistré dans le système de gestion du réseau de train pour permettre au système de diagnostic de l'extraire.

8 Gestion de Réseau de Train

8.1 Généralités

8.1.1 Contenu du présent article

La Gestion de Réseau de Train spécifie un certain nombre de services d'aide à la mise en service, aux essais, à l'exploitation et à l'entretien d'un Réseau Embarqué de Train, tel que:

a) l'identification et le contrôle des stations;

b) la gestion des couches de liaison du bus de train et du Réseau de Rame;

c) la distribution du routage et de la topographie;

d) la lecture à distance et le forçage de variables;

e) le téléchargement.

Tous ces services peuvent être demandés à distance par un processus gestionnaire.

Ces services sont exécutés par chaque station à travers un processus d'Agent. Le présent article spécifie les objets gérés localement et la manière dont les interfaces de l'agent y accèdent.

Le présent article spécifie les messages de gestion échangés entre le Gestionnaire et l'Agent pour la gestion de réseau.

En outre, le présent article spécifie la manière dont les services définis par l'utilisateur peuvent accéder à l'Agent.

NOTE 1 La Gestion de Réseau de Train ne spécifie pas les messages spécifiques à l'application (par exemple, diagnostique de train conformément au CODE UIC 557 ou l'identification des rames conformément au CODE UIC 556). Toutefois, les applications utilisateur peuvent utiliser les services de gestion pendant le fonctionnement normal.

NOTE 2 La Gestion de Réseau de Train ne tient pas compte des services de gestion directement liés à l'application. En particulier, la Description d'Équipement Virtuel, qui s'applique au véritable rôle de l'équipement (la climatisation, par exemple), ne fait pas partie de cet article.

8.1.2 Structure du présent article

Le présent article est structuré comme suit:

Paragraphe 8.1 - Généralités

Paragraphe 8.2 – Gestionnaire, Agents et interfaces

Paragraphe 8.3 – Objets gérés

Pour chaque objet, il est spécifié:

- la description de l'objet;
- l'accès à l'objet;
- les services offerts par l'objet.

Paragraphe 8.4 – Services et messages de gestion

Pour chaque service, il est spécifié:

- sa description;
- paramètres et Call_Message;
- paramètres et Reply_Message.

Paragraphe 8.5 – Procédures d'interface

Pour chaque interface sont spécifiées les procédures d'accès à l'Agent et son contrôle local.

8.2 Gestionnaire, agents et interfaces

8.2.1 Gestionnaire et agent

Les services de Gestion du Réseau doivent être fournis dans chaque Station par un agent.

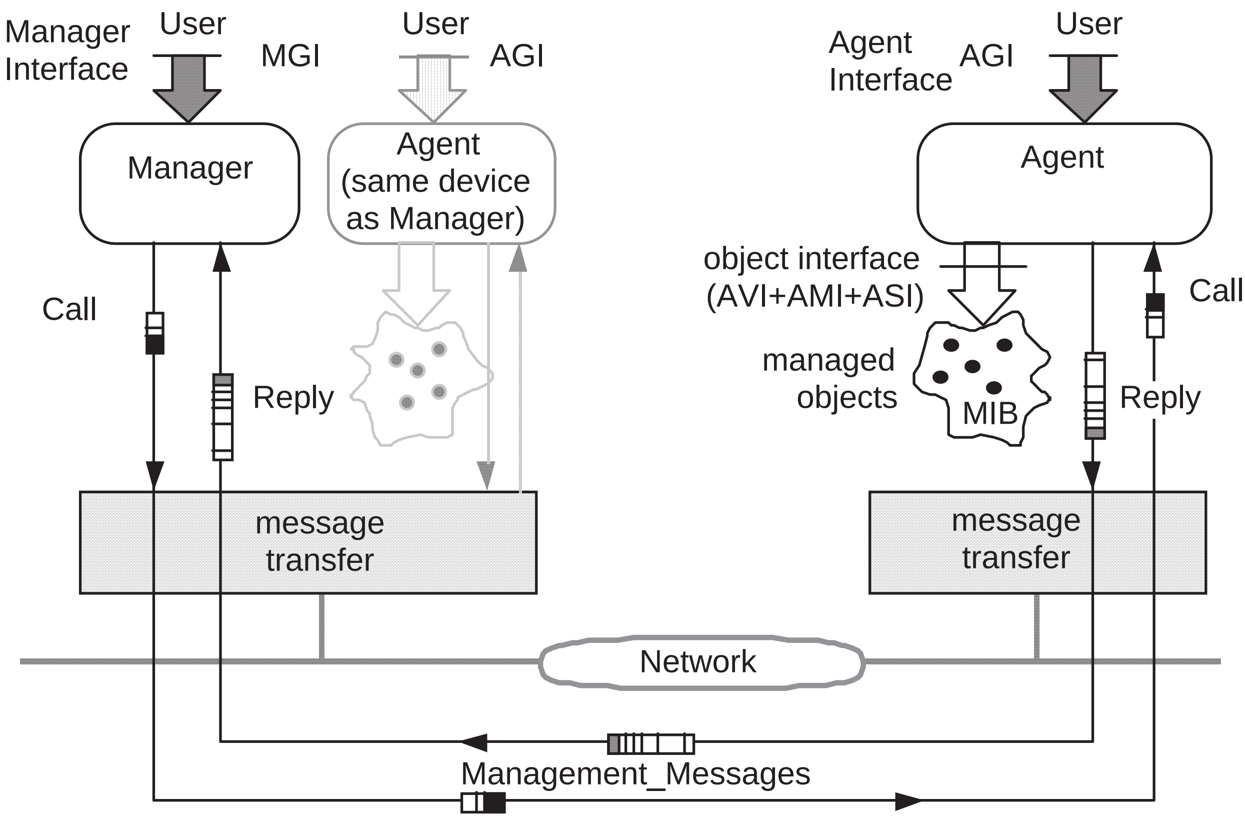
L'Agent doit être identifié par le Station_Id de la Station dans laquelle il réside.

Les services de Gestion de Réseau doivent être demandés par un Gestionnaire.

8.2.2 Protocole des messages de gestion

Pour la Gestion du Réseau, le Gestionnaire et les Agents communiquent sur le réseau en échangeant des messages de gestion, à l'aide des services de messagerie du Réseau Embarqué de Train (voir la Figure 154).

Le Gestionnaire doit exercer les fonctions d'Appelant et l'Agent celles du Répondeur.



Légende

Anglais	Français
Manager Interface	Interface du Gestionnaire
User	Utilisateur
Agent Interface	Interface de l'agent
Manager	Gestionnaire
Agent (same device as Manager)	Agent (même dispositif que le Gestionnaire)
object interface	interface objet
Call	Appel
Reply	Réponse
managed objects	objets gérés
message transfer	transfert de message
Network	Réseau

Figure 154 – Messages de gestion

Le Gestionnaire accède à un objet distant en deux étapes:

a) il envoie un Call_Message de gestion;

b) l'Agent décode le message, accède à l'objet en question et renvoie un Reply_Message de gestion, avec le résultat du service.

L’Agent doit être accessible via une Adresse Système et une Adresse Utilisateur, avec Function_Id = 253.

Le Gestionnaire doit être accessible à la fois via une Adresse Système et une Adresse Utilisateur, avec Function_Id = 254.

Le format des messages de gestion doit être tel que défini dans les paragraphes suivants.

8.2.3 Interfaces

8.2.3.1 Interface d’objet

Les objets de communication se rapportent à la communication de réseau, contrairement aux autres objets, qui se rapportent à d’autres propriétés d’une Station.

EXEMPLE 1 La configuration d’un Administrateur de Bus est un objet de communication.

EXEMPLE 2 Les régions ou domaines de mémoire, l’échéancier, l’horloge, la configuration de Station et les descripteurs ne sont pas des objets de communication.

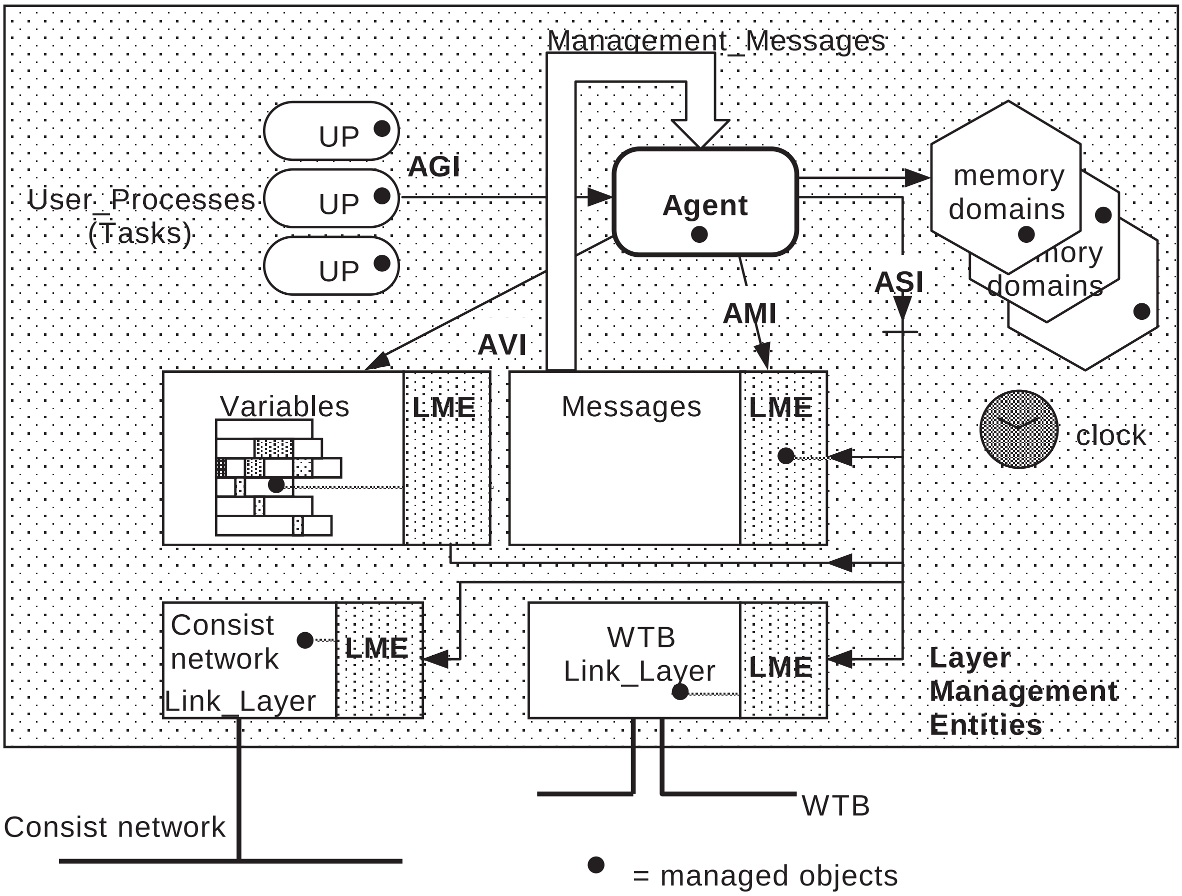
L’Agent doit accéder aux objets de communication par les interfaces définies pour l’accès général dans la présente Norme, et en particulier à travers les interfaces:

a) AVI (Application_Variables_Interface) pour les Variables;
b) AMI (Application_Messages_Interface) pour les Messages;
c) ASI (Application_Supervisory_Interface) pour les objets non accessibles par des processus d’utilisateur.

L'interface ASI fournit l’accès aux objets situés dans les différentes couches de communication. L’Agent accède à ces objets à travers l’Interface de Gestion de Couche de la couche dans laquelle ils résident (Figure 155). La définition de cette interface est incluse dans les articles qui spécifient la couche correspondante: Article 6 (RTP), CEI 61375-3-1 (MVB) et le présent document (WTB).

NOTE L’entité qui accède effectivement aux objets dans chaque couche est appelée Entité de Gestion de Couche (ou LME).

L’Agent doit également pouvoir accéder aux objets de non-communication. L’interface le permettant n’est pas spécifiée.



Légende

Anglais	Français
memory domains	domaines de mémoire
clock	horloge
Consist network	Réseau de rame
Layer Management Entities	Entités de Gestion de Couche
managed objects	objets gérés

Figure 155 – Interface de l'Agent sur une Station (passerelle)

8.2.3.2 Interface du Gestionnaire (MGI)

Le Gestionnaire doit offrir un ensemble de procédures pour accéder aux objets distants, qui forment l'Interface du Gestionnaire, ou MGI (voir la Figure 154).

A chaque service de l'Interface du Gestionnaire doit correspondre un Call_Message de gestion envoyé par le Gestionnaire, auquel l'Agent doit répondre par un Reply_Message de gestion.

Les procédures de la MGI ne sont pas spécifiées individuellement, mais deux procédures génériques fournissant tous les services sont définies.

Le format du Message de Gestion doit également être utilisé pour les paramètres de la procédure de service du Gestionnaire.

8.2.3.3 Interface de l'agent (AGI)

Certains objets gérés doivent accéder à l'Agent. C'est le cas pour les tâches qui nécessitent de demander l'état de l'Agent pour la synchronisation ou pour rendre compte de changements d'état.

A cet effet, l'Agent doit fournir une interface appelée Interface de l'Agent (AGI), laquelle permet aux utilisateurs locaux de lire et modifier directement certains objets gérés (voir 8.5.2).

NOTE Aucun message de réseau n'est associé à l'interface de l'agent.

8.3 Objets Gérés

8.3.1 Attributs d'objet

Chaque objet géré doit comporter quatre attributs:

a) un identifiant d'objet, une chaîne en texte clair de 31 caractères au maximum identifiant l'objet;
b) un état (en lecture seule) indiquant l'état de l'objet;
c) un contrôle (en écriture seulement) agissant sur l'objet;
d) une partie de paramètres (lecture ou écriture), contenant des parties modifiables de l'objet, dans la mesure où ils ne sont pas déjà contenus dans le statut ou le contrôle.

Les attributs d'un objet doivent être traités par les services définis dans le présent article ou par les services définis par l'utilisateur.

Chaque service défini pour un objet doit être décrit par une information textuelle, appelée Service_Descriptor. Pour les services standards, les informations textuelles peuvent contenir une référence à la présente Norme. Pour les services définis par l'utilisateur, les informations textuelles ne sont pas spécifiées.

8.3.2 Objets de la Station

8.3.2.1 Objet Station_Status

Chaque Station doit mettre en œuvre un objet Station_Status, indiquant les caractéristiques générales d'une Station et certains paramètres dynamiques.

Quand une Station est connectée à un seul Réseau de Rame et à aucun bus de train, le Station_Status doit être une copie du Device_Status du Réseau de Rame. Sinon, l'état des différentes couches de liaison doit être combiné en OU pour former le Station_ Status.

Le format du Station_Status doit se présenter de la manière suivante (voir la Figure 156):

```
Station_Status::=       RECORD
  {
  station_capabilities  BITSET4
    {                                         -- capacités de base d'une
                                                 station
    sp    (0),                                -- dispositif spécial
    ba    (1),                                -- Administrateur du MVB
    gw    (2),                                -- passerelle ou noeud du bus
                                                 de train
    md    (3)                                 -- services de messagerie
    },
  class_specific        WORD4,                -- toujours zéro
  common_flags          BITSET8
    {
    lat    (0),                               -- inutilisé = 0
    lnd    (1),                               -- liaison perturbée
    ssd    (2),                               -- dérangement du processus
                                                 contrôlé
                                                 (par exemple perte
                                                 d'alimentation)
    sdd    (3),                               -- Dérangement du dispositif:
                                                 (erreur de somme de
                                                 contrôle, par exemple)
    scd    (4),                               -- Dérangement de la
                                                 communication:
                                                 activé quand le
                                                 temporisateur de supervision
                                                 d'un port quelconque se
                                                 déclenche dans un quelconque
                                                 Traffic_Store du dispositif,
                                                 désactivé quand tous les
                                                 ports fonctionnent
                                                 normalement.
    frc    (5),                               -- station forcée: un port de
                                                 ce dispositif a été forcé
    snr    (6),                               -- station hors-service
                                                 (par exemple non
                                                 initialisée)
    ser    (7)                                -- station réservée par un
                                                 gestionnaire.
    }
  }
```

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>sp</td><td>ba</td><td>gw</td><td>md</td><td></td><td></td><td></td><td></td><td>lat</td><td>lnd</td><td>ssd</td><td>Sdd</td><td>scd</td><td>frc</td><td>snr</td><td>ser</td></tr>
<tr><td colspan="4">station_capabilities</td><td colspan="4">class_specific</td><td colspan="8">Common_flags</td></tr>
</table>

Figure 156 – Station_Status

8.3.2.2 Objet Station_Control

Chaque Station doit mettre en œuvre un objet Station_Control avec deux services associés:

a) démarrer une Station et attribuer les paramètres de configuration (le nom de la Station, par exemple);

b) redémarrer une Station, c'est-à-dire arrêter toutes les tâches, vider tous les tableaux, fermer toutes les conversations et redémarrer uniquement le Messager et l'Agent.

Si seules des tâches d'application doivent être démarrées ou arrêtées, l'objet de tâche doit permettre un redémarrage en douceur.

8.3.2.3 Objet d'inventaire de station

Chaque Station doit mettre en œuvre un objet d'inventaire en lecture seule décrivant les caractéristiques statiques d'une Station, comportant:

a) son identification:
   - le fabricant,
   - le numéro de série,
   - le nom et le Station_Identifier;
b) ses capacités:
   - la version des logiciels,
   - les services pris en charge,
   - la liste des couches de liaison.

La liste des couches de liaison est représentée par un Link_Set, un bitset de 16 bits comportant un bit pour chacune des couches de liaison. A chaque couche de liaison est associé un Traffic_Store. Le Link_Set est défini de la manière suivante:

```
Link_Set::=            BITSET16
  {
  link_layer15  (15),  --                   premier bit transmis
  link_layer14  (14),
  link_layer13  (13),
  link_layer12  (12),
  link_layer11  (11),
  link_layer10  (10),
  link_layer9    (9),
  link_layer8    (8),
  link_layer7    (7),
  link_layer6    (6),
  link_layer5    (5),
  link_layer4    (4),
  link_layer3    (3),
  link_layer2    (2),
  link_layer1    (1),
  link_layer0    (0)
  }
```

8.3.2.4 Objet de réservation de station

Chaque Station doit mettre en œuvre un objet de réservation permettant à un Gestionnaire de réserver cette Station pour son usage exclusif.

L'objet de réservation doit être un sémaphore doté d'un temporisateur assurant l'accès exclusif aux objets modifiables.

L'état de ce sémaphore doit être reflété dans le bit 'ser' du Station_Status ("1" = réservé, "0" = non réservé).

Les services sur l'objet de réservation sont "réserve" (reserve) et "libère" (release).

Un Gestionnaire doit réserver une Station avant de modifier ses objets gérés.

Un Gestionnaire qui projette de réserver une Station doit envoyer à l’Agent une Demande de Réservation.

Un Agent qui accepte une réservation doit définir l’objet de réservation et enregistrer l’identité de son Gestionnaire.

Le propriétaire de l’objet de réservation doit être identifié par l’adresse de l’appelant de 24 bits reçue par l’Agent.

De plus, l’appel de réservation fournit un mot de 32 bits pour identifier plus exactement le Gestionnaire (ou la personne qui gère le réseau) pour des applications spécifiques.

Si un Gestionnaire réserve plusieurs stations, il doit le faire par ordre croissant de leur Application_Address pour prévenir des blocages avec un autre Gestionnaire.

Un Agent doit rejeter tout appel de service de modification, s’il n’a pas été réservé (bit ‘ser’ = 0) ou si la demande est émise par un autre Gestionnaire non enregistré.

Le temporisateur assure qu’une Station ne reste pas bloquée si le Gestionnaire se retire sans libérer la Station.

Le temporisateur du sémaphore doit être redémarré chaque fois que l’Agent reçoit un Call_Message de gestion de la part de son Gestionnaire enregistré.

Le sémaphore de réservation doit être purgé:

a) par une inauguration du bus de train;
b) si la station est réinitialisée;
c) par un message de réservation "outrepasse" (override).

NOTE 1 Les services non modificateurs (lecture) ne sont pas soumis à réservation. Ils peuvent être appelés par n’importe quel Gestionnaire ou par une autre fonction. Seul un Gestionnaire enregistré peut émettre des demandes modificatrices (écriture).

NOTE 2 L’identité du Gestionnaire et de l’Agent peut être affectée par une inauguration.

NOTE 3 Le message "outrepasse" est une méthode de dernier recours pour réserver une Station qui reste bloquée dans un état réservé. Il convient de l'utiliser avec précaution.

NOTE 4 La réservation n’assure pas la sécurité d’accès. Elle peut uniquement protéger contre certaines fausses manœuvres. Si un accès de sécurité est nécessaire, il est recommandé d’utiliser des mots de passe au niveau de la station de travail.

8.3.3 Objets de liaison WTB

8.3.3.1 Objet WTB_Status

Chaque nœud associé au Bus de Train doit mettre en œuvre un objet WTB_Status en lecture seule spécifié dans la présente Norme (Bus de Train Filaire).

L’objet WTB_Status identifie la version du matériel et du logiciel, définit les paramètres statiques et dynamiques, révèle les erreurs et les statistiques de la Link_Layer du WTB et contient le mot d’Etat du Nœud (Node Status) correspondant à cette connexion WTB.

L’Agent accède à cet objet par l’interface de supervision de couche de liaison du WTB, ls_t_xxx.

L’objet WTB_Status comprend les structures de donnée suivantes, fournies par l’application locale et définies dans la présente Norme:

a) Node_Descriptor, structure définissant les trames de Données de Processus, (voir 5.5.2.1);
b) Node_Report, un BITSET8 signalant des dérangements, (voir 5.5.2.2);
c) User_Report, un WORD8 fourni par l'application (voir 5.5.2.3);
d) Node_Key, un RECORD contenant le Node_Type et la Node_Version (voir 5.5.2.4);
e) Ls_T_State, un ENUM8 indiquant l'état d'un nœud (voir 5.6.4.2.2);
f) Node_Strength, un ENUM8 exprimant la force du nœud (voir 5.6.4.16.3).

8.3.3.2 Objet WTB_Topography

Chaque nœud associé au Bus de Train doit mettre en œuvre un objet WTB_Topography en lecture seule spécifié dans la présente Norme (Bus de Train Filaire).

L'Agent accède à cet objet par l'interface de gestion de liaison, ls_t_xxx.

NOTE 1 La Topographie WTB décrit la configuration courante du bus de train, énumère les nœuds existants et leurs descripteurs, et rend compte de l'état du maître.

NOTE 2 Il convient que toutes les applications qui envoient des messages via le bus de train aient accès à l'objet Topography, puisque la topographie assure la cohérence de l'adressage quand la composition change (voir 6.3).

NOTE 3 La topographie peut être obtenue de n'importe quelle Station. Toutefois, seule une Station dotée d'une connexion de Bus de Train peut donner la version actuelle de la topographie.

NOTE 4 L'objet de topographie se compose d'une partie obligatoire et d'une partie Inauguration_Data fournie par l'utilisateur.

8.3.4 Objets variables

8.3.4.1 Objet de configuration de port

Chaque Station ayant la capacité Process_Data doit mettre en œuvre l'objet Ports_ Configuration en lecture seule, lequel indique le nombre de ports et leurs adresses.

Un port peut être de taille fixe (Réseau de Rame) ou de taille variable (WTB).

La taille d'un port est déterminée par son F_code, comme suit:

```
F_Code::=             ENUM4
  {
  PD16    (0),                               -- port de 16 bits
  PD32    (1),                               -- port de 32 bits
  PD64    (2),                               -- port de 64 bits
  PD128   (3),                               -- port de 128 bits
  PD256   (4),                               -- port de 256 bits
  PD512   (5),                               -- port de 512 bits
  PD1024  (6),                               -- port de 1024 bits
  PDW     (7)                                -- port de taille variable
                                                (WTB)
  }
```

Un port est défini sur un bus donné par son adresse de port de 12 bits. Sur le WTB, seuls les 6 bits de poids faible sont utilisés. Le F_code et l'adresse d'un port sont combinés pour former une structure de 16 bits appelée FcodeAdr:

```
FcodeAdr::=            RECORD
  {
  f_code               F_Code,            -- F_code indiquant la taille
                                             du port
  adresse              UNSIGNED12         -- adresse logique
  }
```

Chaque port est caractérisé par les attributs suivants:

```
Port_Object::= RECORD
  {
  f_code               F_Code             -- F_code du port
  port_address         UNSIGNED12         -- adresse du port
  port_config          BITSET4
    {
    src    [0]                            -- port activé en tant
                                             qu'éditeur (émetteur)
    snk    [1]                            -- port activé en tant que
                                             souscripteur (destinataire)
    twc    [2]                            -- trames de transfert de port
                                             avec somme de contrôle
    frc    [3]                            -- port forcé à une valeur
    }
  port_size            UNSIGNED8          -- taille maximale du port en
                                             octets, pour ports PDW (WTB)
  }
```

8.3.4.2 Objet Variables

8.3.4.2.1 Objet

Chaque Station avec la capacité Process_Data doit mettre en œuvre l'objet Variables, indiquant la valeur, la validité et le niveau de rafraîchissement d'une Variable.

NOTE La période individuelle avec laquelle un port est rafraîchi est contrôlée par l'objet Bus_Configuration du Réseau de Rame ou par le contrôle d'accès du support sur nœud du Bus de Train. L'association entre les ports, les variables et les adresses de bus est une question d'application.

8.3.4.2.2 Accès

Comme tout autre processus d'application, l'Agent accède aux Variables par l'interface AVI. Cette interface permet la lecture et le forçage des variables dans le Traffic_Store, individuellement ou par grappes (Clusters).

8.3.4.2.3 Services

Tous les services à distance sur les variables sont exécutés sur les grappes (variables multiples), l'accès à une variable individuelle étant un cas spécial.

Les services suivants accèdent à distance aux variables:

a) lire: récupère la valeur d'une variable, sa variable de contrôle et son niveau de rafraîchissement;
b) forcer: impose une valeur spécifiée à une variable. Cette valeur ne peut pas être écrasée par une application (émetteur) ou par le bus (destinataire) tant que le forçage de la variable n'est pas levé. Les autres variables du même Dataset peuvent être forcées ou non:
   - une variable forcée ne peut pas être écrasée par son producteur, l'Agent doit demander la permission de la modifier à travers l'AGI;

- si la variable se trouve dans un port émetteur, le bus diffuse sa valeur à tous les destinataires à partir de la période suivante.
- si la variable se trouve dans un port destinataire, la valeur forcée est seulement visible à sa propre station. Le bus ne peut écrire cette variable tant que la variable est forcée, mais le rafraîchissement reste toujours commandé par le bus.
- si une variable dans une station a été forcée, l'Agent active le bit "frc" dans son Station_Statut et dans le Device_Status du Réseau de Rame connecté au Traffic_Store forcé;

c) Lever le forçage: libère une variable forcée pour que son application puisse de nouveau modifier sa valeur;

d) Unforce_all: libère toutes les variables dans un Traffic_Store;

e) read_bindings: indique à quel port est liée une variable locale;

f) write_bindings: lie une variable locale à un port.

NOTE 1 Un processus utilisateur peut accéder à une variable en demandant à son agent local d'exécuter une read_variables ou une force_variables sur l'un de ses propres Traffic_Stores. Toutefois, cela prend beaucoup plus de temps que l'accès direct.

NOTE 2 Une application consommateur qui a besoin de distinguer une variable forcée d'une variable non forcée peut attribuer la valeur '10'B à la variable de contrôle d'une variable forcée.

NOTE 3 Il convient d'appeler une fois Unforce_all pour chaque Traffic_Store tant que le bit 'frc' du Station_Status n'est pas réinitialisé.

8.3.4.3 Port_Attach

Chaque dispositif MVB de Classe 2 doit mettre en œuvre l'objet Port_Attach, lequel définit le triage entre les ports et les points d'entrée/sortie. Cet objet peut être seulement modifié à distance.

8.3.5 Objets du Messager

8.3.5.1 Etat du Messager

Chaque Station fournissant le service de Messagerie doit mettre en œuvre l'objet d'état du Messager en lecture seule, lequel rend compte de la version du logiciel, des statistiques de configuration et d'utilisation des deux protocoles:

a) le protocole de diffusion unique; et

b) le protocole de diffusion (facultatif).

L'Agent accède à cet objet par l'interface ASI du Messager.

8.3.5.2 Contrôle du Messager

Chaque Station fournissant le service de Messagerie doit mettre en œuvre l'objet de contrôle du Messager, qui permet de configurer à distance les paramètres de communication par messages.

8.3.5.3 Objets du répertoire

Chaque Station fournissant le service de Messagerie doit mettre en œuvre le Function_Directory, qui indique, pour chaque Fonction, la Station qui l'exécute.

Chaque Station ayant la capacité de messagerie, mais n'utilisant pas le routage simple, doit mettre en œuvre le Station_Directory, qui met en correspondance le Station_Identifier et la Next_Station et son adresse physique (Bus_Id et Device_Address).

Chaque Station terminale prenant en charge le protocole de distribution et chaque Station Routeur doit mettre en œuvre l'objet Group_Directory, qui indique à la Station les groupes auxquels elle appartient.

Chaque Nœud d'un bus de train à configuration fixe ayant la capacité de messagerie mais n'utilisant pas le simple routage, doit mettre en œuvre le Node_Directory, qui indique à quelle adresse de dispositif correspond une adresse de nœud donnée.

Il doit exister deux services pour chaque répertoire:

a) read_xxx_directory, qui lit le répertoire dans sa totalité;
b) write_xxx_directory, qui écrit le répertoire dans sa totalité (en l'ayant au préalable purgé ou non).

NOTE 1 Le Station_Directory peut être remplacé par un routage simple dans les stations terminales, auquel cas aucun objet Station_Directory n'est fourni.

NOTE 2 L'Agent accède aux répertoires par l'interface AMI, comme dans toute autre application.

8.3.6 Objets de domaine

8.3.6.1 Objets

Chaque Station ayant la capacité de téléchargement des zones de mémoire doit mettre en œuvre l'objet Domaine.

Le nombre de domaines par station n'est pas spécifié.

Pour des raisons technologiques, il convient qu'un domaine soit entièrement contenu dans le même type de mémoire.

Comme cas particulier, le Gestionnaire peut accéder aux zones de mémoire physiques d'un domaine.

NOTE 1 Les domaines sont des zones de mémoire contenant un code ou des données. Les domaines peuvent se trouver indifféremment dans des mémoires de différentes technologies, RAM, EEPROM ou Flash-EPROM.

NOTE 2 L'Agent accède aux domaines par des procédures spécifiques à la mise en œuvre, qui assurent la lecture/écriture des emplacements de mémoire de technologies différentes, pouvant être protégés par des droits d'accès ou par la gestion de la mémoire.

NOTE 3 L'Accès aux domaines exige une connaissance précise de la configuration d'une Station.

8.3.6.2 Services

Les services sur les objets Domaine doivent comprendre:

a) la configuration du chargement (préparer le téléchargement, vérifier, amorcer);
b) le chargement du segment;
c) la lecture de mémoire;
d) l'écriture de mémoire.

NOTE 1 Le chargement du Domaine peut écraser des zones existantes ou le logiciel de communication et l'Agent lui-même.

NOTE 2 Avant de télécharger un domaine, il est plus sûr pour le Gestionnaire d'émettre auparavant un appel "reset_station". Dans ce cas, il convient que le Gestionnaire réserve la Station une fois de plus pour le téléchargement.

8.3.7 Objets de tâche

8.3.7.1 Objets

Chaque Station capable de gérer des tâches utilisateur doit mettre en œuvre l'objet Tâche en écriture seulement.

Le nombre de tâches n'est pas spécifié.

Les tâches sont traitées comme un ensemble pour les besoins de gestion.

L'objet de contrôle de tâche commande l'exécution de toutes les tâches.

8.3.7.2 Accès

L'Agent est supposé accéder à l'échéancier pour démarrer et arrêter des tâches.

L'accès aux tâches est dépendant de la mise en œuvre.

8.3.7.3 Services

Trois services de tâche sont spécifiés:

a) le démarrage, qui commence toutes les tâches;
b) l'arrêt, qui arrête toutes les tâches;
c) tasks_list, qui lit la liste des tâches installées sur la station ainsi que leur état.

Avant d'arrêter une tâche, l'Agent doit obtenir la permission de la tâche par l'AGI (voir 8.2.3.3), sauf quand l'accès "outrepasse" est spécifié.

8.3.8 Objet d'horloge

8.3.8.1 Objets

Chaque Station dotée d'une horloge accessible à distance doit exécuter l'objet Horloge.

La précision de l'horloge ne fait l'objet d'aucune exigence.

NOTE 1 La mise à l'heure ou la lecture de l'horloge par un Management_Message est assujettie à des délais de transport imprévisibles. Plus particulièrement, une limite supérieure pour la livraison d'un message ne peut pas être donnée puisqu'elle dépend de la présence d'autres messages dans la file d'attente de cette station et des autres.

NOTE 2 Une mise à l'heure plus précise de l'horloge peut être obtenue en laissant le Bus_Administrator envoyer une variable de synchronisation à un instant déterminé. Cela est une question de mise en œuvre.

8.3.8.2 Services

Deux services d'horloge sont spécifiés:

a) read_clock, qui lit le temps actuel de l'horloge;
b) set_clock, qui met l'horloge à l'heure.

8.3.9 Objet de journal

8.3.9.1 Objet

Une Station peut mettre en œuvre un objet de journal pour les besoins du débogage.

Le journal est composé d'une liste d'entrées définies par l'utilisateur avec un numéro de série et un horodatage. Il est prévu de mettre en œuvre le journal comme un tampon circulaire, dans lequel les dernières entrées "j" sont enregistrées, en écartant les plus anciennes.

Le journal peut être lu par plusieurs gestionnaires, la lecture n'étant pas consommatrice.

Le journal peut être rapidement écrit en parallèle par des applications présentes sur le nœud. Pour éviter la mise en mémoire tampon, un schéma d'index permet de sélectionner la partie valable du journal.

Chaque tâche peut entrer des événements dans le journal par l'intermédiaire de l'interface AGI.

8.3.9.2 Services

Un service de journal est spécifié:

– read_journal, qui lit les dernières entrées du journal.

8.3.10 Objet d'Equipement

8.3.10.1 Objet

Une Station peut mettre en œuvre un descripteur d'équipement qui décrit les équipements pris en charge.

8.3.10.2 Services

Un service est spécifié:

– read_equipment_descriptor, qui récupère un pointeur vers l'emplacement de mémoire où se trouve le descripteur d'équipement.

8.4 Services et messages de gestion

8.4.1 Notation pour tous les messages de gestion

8.4.1.1 Structure d'un message

Chaque service doit être appelé par un échange de message de gestion appel/réponse au format suivant:

```
Management_Message::= RECORD
  {
  tnm_key               ENUM8               -- premier octet
    {
    CALL    ('02'H),                        -- Call (appel)
    REPLY   ('82'H)                         -- Reply (réponse)
},
  message               ONE_OF [tnm_key]    -- sélectionne l'appel ou la
                                               réponse
    {
    [CALL]              Call_Mgt_Message,   -- décrit plus bas
    [REPLY]             Reply_Mgt_Message   -- décrit plus bas
    }
  }
```

Le bit de poids fort de tnm_key doit indiquer s'il s'agit d'un message d'appel ou de réponse.

Pour la gestion de réseau privé, les valeurs de ‘tnm_key’ doivent se trouver dans l’intervalle ‘40’H -’7F’H (appel) ou ‘C0’H – ‘FF’H (réponse). D’autres valeurs sont réservées pour une utilisation ultérieure.

Pour l’attribution par défaut de tnm_keys, le code de compagnie UIC peut être ajouté à ‘40’H pour former le tnm_key.

NOTE Les champs des messages de gestion ont été alignés sur une limite de 32 bits pour accélérer le fonctionnement des processeurs de 32 bits. Cet alignement suppose que le premier champ, ‘tnm_key’, se trouve à une adresse divisible par quatre. Il convient que le service de messagerie respecte cette convention tant pour l’envoi que pour la réception.

8.4.1.2 Notation pour le SIF_code

Le SIF_code indique le service demandé. Le bit de poids faible indique s’il s’agit d’un service de lecture ou d’écriture (modification). Les services de lecture peuvent être utilisés sans réservation préalable. Les SIF_codes suivants sont définis pour la paire tnm_key (‘02’H / ‘82’H) par défaut.

```
Sif_Code::= ENUM8      --                  choix du 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_WTB_USER_REPORTport (25),
  LINE_FAULT_LOCATION_DETECTION (26),
  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) réservés pour
                                              services utilisateur.
  }
```

NOTE Les SIF_codes des services MVB sont inclus pour des raisons d'exhaustivité. Pour obtenir une description des services MVB, voir la CEI 61375-3-1.

8.4.1.3 Notation pour un message de gestion d'appel

```
Call_Mgt_Message::=     RECORD
  {
  sif_code              Sif_Code,           -- le deuxième octet est le
                                               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,
}
  }
```

8.4.1.4 Notation pour un message de gestion de réponse

```
Reply_Mgt_Message::=   RECORD
  {
  sif_code        Sif_Code,                -- le deuxième octet est le
                                              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]            Replay_Read_Equipment
}
  }
```

8.4.1.5 Rapport d'état ou d'erreur

L'Agent doit rendre compte du succès ou de l'échec du service par son état de répondeur, lequel n'est pas transmis dans le corps du message de réponse, mais est passé comme paramètre séparé dans le Session_Header.

Le résultat établi par l'Agent peut être modifié par le réseau en cas d'erreur de communication. Dans ce cas, le réseau renvoie une erreur de communication, en remplaçant le code de l'Agent par son propre Am_Result.

L'état du répondeur doit être renvoyé comme un paramètre de "mm_service_conf" au Gestionnaire.

Le rapport d'état et d'erreur doit combiner les résultats de l'agent et du réseau, comme spécifié par le type Mm_Result:

```
Mm_Result::=               ENUM8
  {
  MM_OK                    (0)      -- achèvement normal
  AM_FAILURE               (1)      -- défaillance non spécifiée
  AM_BUS_ERR               (2)      -- transmission bus pas possible
  AM_REM_CONN_OVF          (3)      -- trop de connexions entrantes
  AM_CONN_TMO_ERR          (4)      -- Connect_Request sans réponse
  AM_SEND_TMO_ERR          (5)      -- temporisation SEND_TMO échue
  AM_REPLY_TMO_ERR         (6)      -- réponse non reçue
  AM_ALIVE_TMO_ERR         (7)      -- temporisation ALIVE_TMO échue
  AM_NO_LOC_MEM_ERR        (8)      -- manque de mémoire ou de
                                       temporisateurs
  AM_NO_REM_MEM_ERR        (9)      -- partenaire manque de mémoire ou de
                                       temporisateurs
  AM_REM_CANC_ERR          (10)     -- annulé par le partenaire
  AM_ALREADY_USED          (11)     -- opération déjà réalisée
  AM_ADDR_FMT_ERR          (12)     -- erreur de format d'adresse
  AM_NO_REPLY_EXP_ERR      (13)     -- réponse non attendue
  AM_NR_OF_CALLS_OVF       (14)     -- trop d'appels demandés
  AM_REPLY_LEN_OVF         (15)     -- Reply_Message trop long
  AM_DUPL_LINK_ERR         (16)     -- erreur de conversation dupliquée
  AM_MY_DEV_UNKNOWN_ERR    (17)     -- mon adresse est inconnue
  AM_NO_READY_INST_ERR     (18)     -- instance du répondeur non prête
  AM_NR_OF_INST_OVF        (19)     -- trop d'instances de répondeur
  AM_CALL_LEN_OVF          (20)     -- Call_Message trop long
  AM_UNKNOWN_DEST_ERR      (21)     -- dispositif du partenaire inconnu
  AM_INAUG_ERR             (22)     -- inauguration du train a eu lieu
  AM_TRY_LATER_ERR         (23)     -- (pour usage interne seulement)
  AM_FIN_NOT_REG_ERR       (24)     -- adresse finale non enregistrée
  AM_GW_FIN_NOT_REG_ERR    (25)     -- adresse finale non enregistrée dans
                                       le routeur
  AM_GW_ORI_REG_ERR        (26)     -- adresse d'origine non enregistrée
                                       dans le routeur

  MM_SIF_NOT_SUPPORTED     (33)     -- SIF_code non pris en charge
  MM_RDONLY_ACCESS         (34)     -- accès en lecture seule autorisée
  MM_CMD_NOT_EXECUTED      (35)     -- échec du service
  MM_DNLD_NO_FLASH         (36)     -- pas de mémoire non volatile à cette
                                       adresse
  MM_DNLD_FLASH_HW_ERR     (37)     -- erreur matérielle durant le
                                       chargement
  MM_BAD_CHECKSUM          (38)     -- somme de contrôle incorrecte pour ce
                                       domaine
  MM_INT_ERROR             (39)     -- erreur interne
  MM_ER_VERS               (40)     -- version erronée
  MM_BUS_HW_BAD            (41)     -- liaison défaillante
  MM_BUS_HW_NO_CONFIG      (42)     -- matériel non configuré
  MM_LP_ERROR              (43)     -- échec de l'accès au Traffic_Store
```

```
  MM_VERSION_CONFLICT      (44)     -- conflit de versions
  }
```

8.4.1.6 Numérotation des bits et codage des données

La notion des messages de gestion est structurée en unités de 16 bits. Une unité de 16 bits peut comporter deux types de 8 bits, un type de 16 bits ou une partie d'un type de 32 bits. Les correspondances suivantes montrent la numérotation des bits dépendant des types.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
UNSIGNED8, BITSET8, ENUM8, WORD8								UNSIGNED8, BITSET8, ENUM8, WORD8							
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
UNSIGNED16, ENUM16, WORD16															
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

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
BITSET16															
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
UNSIGNED32															
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

L'octet avec la numérotation des bits inférieurs doit être transmis en premier.

8.4.2 Services de la station

8.4.2.1 Read_Station_Status

8.4.2.1.1 Description

Ce service lit à distance l'objet Station_Status.

8.4.2.1.2 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
  {}                    --                    aucun paramètre
```

8.4.2.1.3 Reply_Read_Station_Status

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="8">tnm_key</td><td colspan="8">sif_code = 0</td></tr>
<tr><td colspan="8">bus_id</td><td colspan="8">reserved1</td></tr>
<tr><td colspan="16">device_address</td></tr>
<tr><td colspan="16">station_status</td></tr>
</table>

```
Reply_Read_Station_Status::= RECORD
  {
  bus_id               UNSIGNED8 (0..15),  -- indicatif de la liaison (par
                                              exemple MVB, WTB) par
                                              laquelle l'agent a reçu
                                              l'appel. Ce lien ne peut pas
                                              changer pendant toute la
                                              session de gestion
  reserved1            WORD8 (=0),         -- réservé
  device_address       WORD16,             -- adresse de dispositif du bus
                                              par laquelle l'agent a reçu
                                              l'appel
  station_status       Station_Status      -- voir définition
  }
```

NOTE Un Gestionnaire peut accéder à une Station inconnue par sa Device_Address.

8.4.2.2 Write_Station_Control

8.4.2.2.1 Description

Ce service initialise une Station et lui attribue un nom et un Station_Id.

8.4.2.2.2 Call_Write_Station_Control

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="8">tnm_key</td><td colspan="8">sif_code = 1</td></tr>
<tr><td colspan="7">command</td><td>RST</td><td colspan="8">station_id</td></tr>
<tr><td colspan="16">station_name: STRING32</td></tr>
<tr><td colspan="8">(CHARACTER8)</td><td colspan="8">CHARACTER8 or ‘00’H</td></tr>
</table>

```
Call_Write_Station_Control::=RECORD
  {
  command               BITSET8
    {
    rst       (7)                           -- si 1: initialise la Station,
                                               efface tous les tableaux et
                                               le sémaphore de réservation
                                               et ne démarre que le
                                               messager et l’Agent.
                                               si rst = 0: attribue
                                               seulement le nouveau nom et
                                               le Station_Identifier.
    },
  station_id            UNSIGNED8,          -- Station_Id de 8 bits
                                               attribué à cette Station.
                                               Si station_id = 0, le
                                               Station_Identifier n’est pas
                                               modifié
  station_name          STRING32            -- nom attribué à cette
                                               Station. Si la taille de
                                               station_name est nulle, le
                                               nom de la Station n’est pas
                                               modifié
  }
```

8.4.2.2.3 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       -- sif_code = 1
  {
  bus_id                UNSIGNED8 (0..15),  -- lien par lequel l’Agent a
                                               reçu le Call_Message
  reserved1             WORD8 (=0),         -- réservé
  device_address        WORD16              -- adresse par laquelle l’Agent
                                               a reçu le Call_Message
  }
```

8.4.2.3 Read_Station_Inventory

8.4.2.3.1 Description

Ce service lit l’objet d’inventaire.

8.4.2.3.2 Call_Read_Station_Inventory

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

```
Call_Read_Station_Inventory::= RECORD
  {}                    --                  aucun paramètre
```

8.4.2.3.3 Reply_Read_Station_Inventory

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
tnm_key	sif_code = 2
reserved1	
agent_version: STRING32	
(CHARACTER8)	CHARACTER8 or ‘00’H
manufacturer_name: STRING32	
(CHARACTER8)	CHARACTER8 or ‘00’H
device_type: STRING32	
(CHARACTER8)	CHARACTER8 or ‘00’H
service_set: BITSET256	
SV0	SV255
LL0 link_set	LL15
reserved2	station_id
station_name: STRING32	
(CHARACTER8)	CHARACTER8 or ‘00’H
station_status	

```
Reply_Read_Station_Inventory::= RECORD
  {
  reserved1            WORD16 (=0)        -- utilisé pour l'alignement
  agent_version        STRING32           -- version de l’Agent
                                             (référence au présent
                                             article, par exemple)
  manufacturer_name    STRING32           -- nom du fabricant du
                                             dispositif
  device_type          STRING32           -- nom du dispositif et numéro
                                             de série.
  service_set          BITSET256          -- un bit pour chacun des
                                             services.
                                             (la valeur 1 attribuée au
                                             premier bit signifie que le
                                             dispositif prend en charge
                                             le service
                                             Read_Station_Status)
  link_set             Link_Set           -- un bit pour chaque Traffic
                                             Store ou couche de liaison
                                             pris en charge, chaque
                                             couche de liaison
                                             correspondant à un Traffic
                                             Store, le premier bit
                                             correspondant au
                                             Traffic_Store 0
  reserved2            WORD8 (=0),        -- réservé
  station_id           UNSIGNED           -- indicatif de la Station (0
                                             ou ‘FF’H si non défini)
  station_name         STRING32           -- nom donné à cette Station
                                             (initialement vide)
  station_status       Station_Status     -- Mot d'état de la Station
  }
```

NOTE Un Gestionnaire peut accéder à une Station inconnue par sa Device_Address.

8.4.2.4 Write_Station_Reservation

8.4.2.4.1 Description

Ce service accède à l'objet de réservation, qui réserve ou libère la Station.

L'appel de réservation spécifie le Gestionnaire (par son Application_Address), la temporisation de réservation et les droits d'accès. Il inclut un indicatif de Gestionnaire de 32 bits, dont l'utilisation est dépendante de l'application.

Si la Station est déjà réservée et si le Gestionnaire est différent de celui en cours, la Station doit rejeter la demande de réservation.

Une Station réservée doit activer le bit 'ser' dans son Station_Status et dans le Device_Status de chaque MVB auquel il est connecté.

La temporisation de réservation doit être relancée par la réception d'un appel de service provenant du Gestionnaire en charge.

La Station doit libérer la réservation et désactiver les bits 'ser':

a) à la réception d'un appel de "libération", sans l'option de redémarrage (cas normal);
b) à l'expiration de la temporisation de réservation;
c) au changement d'adresse de nœud du Gestionnaire, suite à une inauguration de train;
d) à la réception d'un "réinitialise" (reset);
e) à la réception d'un "outrepasse" (override).

Dans les deux derniers cas, la Station doit exécuter la procédure souscrite par l'application comme "station_restart" (voir 8.5.2.4).

8.4.2.4.2 Call_Write_Station_Reservation

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="8">tnm_key</td><td colspan="8">sif_code = 3</td></tr>
<tr><td colspan="16">command</td></tr>
<tr><td colspan="16">access_type</td></tr>
<tr><td colspan="16">reservation_time_out</td></tr>
<tr><td colspan="16">manager_id</td></tr>
</table>

```
Call_Write_Reservation::=  RECORD
  {
  command          ENUM16
    {
    RESERVE,       (1),                     -- réserve le dispositif pour
                                               ce gestionnaire
    KEEPREL        (2),                     -- libère et effectue les
                                               modifications
    STARTREL       (3)                      -- libère et redémarre.
    },
  access_type      ENUM16
    {
    WRITEREQ       (0),                     -- accès en écriture demandé
    OVERRIDE       (1)                      -- accès outrepasse réservé.
    },
  reservation_time_out  UNSIGNED16,         -- temps durant lequel la
                                               Station reste réservée, en
                                               multiples de 1 s, mais au
                                               plus 3600 s.
  manager_id       UNSIGNED32               -- identifie le Gestionnaire
                                               (l'utilisation de cet
                                               indicatif est dépendante de
                                               la mise en oeuvre).
  }
```

8.4.2.4.3 Reply_Write_Station_Reservation

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 3							
reserved1															
manager_id															

```
Reply_Write_Reservation::= RECORD
  {
  reserved1             WORD16              -- pour l'alignement
  manager_id            UNSIGNED32          -- identifie le Gestionnaire
                                               (l'utilisation de cet
                                               indicatif est dépendante de
                                               la mise en oeuvre).
  }
```

8.4.2.5 Read_Service_Descriptor

8.4.2.5.1 Description

Ce service lit le Service_Descriptor, le texte de description définissant un service ou l'objet auquel accède ce service. Il est en principe utilisé seulement pour les services définis par l'utilisateur.

8.4.2.5.2 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),          -- réservé
  get_sif_code         Sif_Code               -- service à décrire
  }
```

8.4.2.5.3 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),          -- réservé
  get_sif_code           Sif_Code,            -- service à décrire
  reserved2              WORD16 (=0),         -- réservé
  string_size            UNSIGNED16,          -- jusqu’à 65535 caractères
  service_description ARRAY ALIGN16 [string_size] OF
                         CHARACTER8           -- chaîne définie par
                                                 l’utilisateur
  }
```

8.4.2.6 Read_Links_Descriptor

8.4.2.6.1 Description

Ce service lit le Links_Descriptor, le texte de description qui doit décrire chaque couche de liaison présente dans le Station_Inventory.

8.4.2.6.2 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
  {}                     --                   aucun paramètre
```

8.4.2.6.3 Reply_Read_Links_Descriptor

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 6							
nr_links															
links_descriptor: ARRAY [nr_links] OF															
bus_id								link_type							
link_name: STRING32															
								CHARACTER8 or ‘00’H							

```
Reply_Read_Links_Descriptor::= RECORD
  {
  nr_links             UNSIGNED16 (0..15), -- nombre de liaisons prises en
                                              charge
  links_descriptor     ARRAY [nr_links] OF
    {                                      -- liste des liaisons
                                              existantes, comprenant pour
                                              chacune:
    bus_id             UNSIGNED8 (0..15),  -- indicatif de la liaison
    link_type          ENUM8
      {
      LINK_UNKNOWN   (0),                  -- inconnu
      LINK_MVB       (1),                  -- bus MVB
      LINK_WTB       (2),                  -- bus WTB
      LINK_MBX       (3),                  -- boîte aux lettres en mémoire
      LINK_SER       (4)                   -- liaison série
                                           -- autres réservés
      },
    link_name          STRING32            -- nom de la liaison comme
                                              chaîne de caractères
    }
  }
```

8.4.2.7 Write_Links_Descriptor

8.4.2.7.1 Description

Ce service écrit le Links_Descriptor, le texte qui doit décrire chaque couche de liaison présente dans le Station_Inventory.

8.4.2.7.2 Call_Write_Links_Descriptor

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="8">tnm_key</td><td colspan="8">sif_code = 7</td></tr>
<tr><td colspan="16">nr_links</td></tr>
<tr><td colspan="16">links_descriptor: ARRAY [nr_links] OF</td></tr>
<tr><td colspan="8">bus_id</td><td colspan="8">link_type</td></tr>
<tr><td colspan="16">link_name: STRING32</td></tr>
<tr><td colspan="8"></td><td colspan="8">CHARACTER8 or ‘00’H</td></tr>
</table>

```
Call_Write_Links_Descriptor::= RECORD
  {
  nr_links              UNSIGNED16 (0..15), -- nombre de liaisons prises en
                                               charge
  links_descriptor      ARRAY [nr_links] OF
    {                                       -- liste des liaisons
                                               existantes, comprenant pour
                                               chacune:
    bus_id              UNSIGNED8 (0..15),  -- indicatif de la liaison
    link_type           ENUM8
      {
      LINK_UNKNOWN   (0),                   -- inconnu
      LINK_MVB       (1),                   -- bus MVB
      LINK_WTB       (2),                   -- bus WTB
      LINK_MBX       (3),                   -- boîte aux lettres en mémoire
      LINK_SER       (4),                   -- liaison série
      LINK_CAN       (5),                   -- bus CAN
      LINK_ETH       (6)                    -- Ethernet
                                            -- autres réservés
      },
    link_name           STRING32            -- nom de la liaison
    }
  }
```

8.4.2.7.3 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
  {}                                        -- aucun paramètre
```

8.4.3 Services de liaison du WTB

8.4.3.1 Read_Wtb_Status

8.4.3.1.1 Description

Ce service lit l'état d'une couche de liaison du WTB d'un nœud. Si cette Station ne comporte aucune connexion au bus de train, ce service renvoie une erreur et le Reply_Message se compose uniquement de l'en-tête.

Les données renvoyées correspondent aux structures de données Type_WTBStatus et Type_LLStatisticData de 5.6.4.16.3 et 5.6.4.22.3.

NOTE En cas de litige, les définitions de paramètre sont celles de 5.6.4.16.3 et 5.6.4.22.3.

8.4.3.1.2 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)   -- indicatif de la liaison
  reserved1             WORD8 (=0)          -- réservé
  }
```

8.4.3.1.3 Reply_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	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,          -- incrémenté pour chaque trame
                                                émise
  received_count        UNSIGNED32,          -- incrémenté pour chaque trame
                                                reçue
  frame_errors_count    UNSIGNED16,          -- incrémenté pour chaque trame
                                                erronée
  frame_timeouts_count   UNSIGNED16,         -- incrémenté pour chaque trame
                                                avec temporisation échue
  }

Reply_Read_Wtb_Status::= RECORD
  {
  bus_id                UNSIGNED8 (0..15)    -- indicatif de la liaison
  node_address          WORD8                -- Node_Address (127 si non
                                                nommé)
  wtb_hardware_id       UNSIGNED8            -- indicatif du matériel
  wtb_software_id       UNSIGNED8            -- indicatif de la version du
                                                logiciel de couche de
                                                liaison
  hardware_state        ENUM8
    {
    LS_OK               (0),                 -- WTB_LS_OK     opération
                                                correcte
    LS_FAIL             (1)                  -- WTB_LS_FAIL  défaillance
                                                matérielle
    },
  link_layer_state      Ls_T_State,          -- état principal dans lequel
                                                le noeud se trouve
  net_inhibit           ENUM8                -- 1: un noeud empêche
                                                l'inauguration
  node_address          UNSIGNED8            -- adresse du noeud attribuée
  node_orient           ENUM8                -- orientation du noeud par
                                                rapport au noeud maître
    {
    L_UNKNOWN           (0),                 -- WTB_LS_UNKNOWN
    L_SAME              (1),                 -- WTB_LS_SAME
    L_INVERSE           (2)                  -- WTB_LS_INVERSE
    },
  node_strength         ENUM8                -- force du noeud
    {
    L_UNDEFINED         (0)                  -- force du noeud non définie
    L_SLAVE             (1)                  -- noeud est un esclave
                                                uniquement
    L_STRONG            (2)                  -- noeud est un maître fort
    L_WEAK              (3)                  -- noeud est un maître faible
    },
  node_frame_size       UNSIGNED8,           -- taille de la trame de
                                                Données de Processus émise
                                                par le noeud.
  node_period           UNSIGNED8,           -- période individuelle désirée
                                                en multiples de T_bp
  node_type             UNSIGNED8,           -- descripteur des capacités du
                                                noeud (fait partie de
                                                Node_Key)
  node_version          UNSIGNED8,           -- descripteur des capacités du
                                                noeud (fait partie de
                                                Node_Key)
```

```
  node_report          Node_Report,          -- rapport de noeud à 8 bits,
                                                comme exprimé dans
                                                Status_Response
  user_report          User_Report,          -- rapport utilisateur à
                                                8 bits, comme exprimé dans
                                                Status_Response
                                             -- la structure suivante
                                                correspond au
                                                Type_LLStatisticData
  basic_period_count   UNSIGNED32            -- incrémenté pour chaque
                                                Basic_Period
  inauguration_count   UNSIGNED16            -- incrémenté pour chaque
                                                inauguration
  topography_count     UNSIGNED16            -- incrémenté pour chaque
                                                nouvelle topographie
  transmitted_md_count UNSIGNED32,           -- incrémenté pour chaque
                                                Message_Data_Response
                                                envoyée
  received_md_count    UNSIGNED32,           -- incrémenté pour chaque
                                                Message_Data_Response reçue
  line_status_a1       Type_LineStatus,      -- statistiques pour A1, voir
                                                définition du type
  line_status_a2       Type_LineStatus,      -- statistiques pour A2, voir
                                                définition du type
  line_status_b1       Type_LineStatus,      -- statistiques pour B1, voir
                                                définition du type
  line_status_b2       Type_LineStatus,      -- statistiques pour B2, voir
                                                définition du type
  line_switch_count    UNSIGNED32            -- incrémenté pour chaque
                                                commutation de ligne
}
```

8.4.3.2 Write_Wtb_Control

8.4.3.2.1 Description

Définit les paramètres d'un nœud de Bus de Train.

8.4.3.2.2 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	csl	slp	alw	inh	rmv	snm	stm	wkm	slv	cnf	rst

```
Call_Write_Wtb_Control::= RECORD
  {
  bus_id             UNSIGNED8 (0..15),     -- indicatif de la liaison
  reserved1          WORD8 (=0)             -- réservé
  command            BITSET16
    {
    rsv1       (0),                         -- réservé
    rsv2       (1),                         -- réservé
    rsv3       (2),                         -- réservé
    rsv4       (3),                         -- réservé
    rsv5       (4),                         -- réservé
    csl        (5),                         -- annule la mise en veille
    slp        (6),                         -- met en veille
    alw        (7),                         -- autorise
    inh        (8),                         -- bloque
```

```
    rmv       (9),                           -- retire
    snm       (10),                          -- commence la dénomination
    stm       (11),                          -- met à l’état maître fort
    wkm       (12),                          -- met à l’état maître faible
    slv       (13),                          -- met à l’état esclave
    cnf       (14),                          -- configure
    rst       (15)                           -- initialise noeud
    }
  }
```

8.4.3.2.3 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),  -- indicatif de la liaison
  reserved1            WORD8 (=0)          -- réservé
  }
```

8.4.3.3 Read_Wtb_Nodes

8.4.3.3.1 Description

Récupère la liste des nœuds que le maître a trouvés sur le bus, avec leur Node_Status_Word (cette commande est toujours envoyée au nœud 01).

8.4.3.3.2 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)   -- indicatif de la liaison
  reserved1            WORD8 (=0)          -- réservé
  }
```

8.4.3.3.3 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),  -- indicatif de la liaison
  node_address         UNSIGNED8,          -- adresse de ce noeud (127 si
                                              non nommé)
  reserved1            WORD8 (=0),         -- réservé
  nr_nodes             UNSIGNED8,          -- nombre de noeuds de la
                                              composition.
  bottom_node          UNSIGNED8,          -- Node_Address du noeud avec
                                              l'adresse la plus basse.
  top_node             UNSIGNED8,          -- Node_Address du noeud avec
                                              l'adresse la plus haute.
  node_status_list     ARRAY [MAX_NODES] OF
    {                                      -- liste des états de noeud,
                                              commençant par le noeud le
                                              plus bas, dans l'ordre
                                              d'emplacement des noeuds, et
                                              se terminant avec le noeud
                                              le plus haut, avec pour
                                              chaque noeud:
    node_report        Node_Report         -- Node_Report des différents
                                              noeuds
    user_report        User_Report         -- User_Report des différents
                                              noeuds
    }
  }
```

8.4.3.4 Read_Wtb_Topography

8.4.3.4.1 Description

Lit la topographie. Le format de la Topographie est spécifié dans la liste des paramètres, car il y a de petites différences entre les informations de Topographie transmises à travers le WTB, celles disponibles à l'interface de supervision WTB et celles disponibles à travers la gestion du réseau.

8.4.3.4.2 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),  -- indicatif de la liaison
  reserved1            WORD8 (=0)          -- réservé
  }
```

8.4.3.4.3 Reply_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								node_address							
node_orient								rsv1	rsv2	topo_counter					
individual_period								is_strong							
number_of_nodes								bottom_address							
top_address								reserved1							
inaug_data_max_size								nr_descriptors							
node_descriptions ARRAY [nr_descriptors] OF															
node_type								node_version							
sam	rsv1	node_address						inauguration_data_size							
inauguration_data ARRAY ALIGNED16 [inauguration_data_size] OF															
WORD8								WORD8							

```
Reply_Read_Wtb_Topography::= RECORD
  {
  bus_id                UNSIGNED8 (0..15),  -- indicatif de la liaison
  node_address          UNSIGNED8,          -- Node_Address à laquelle
                                               cette station est reliée
  node_orient           ENUM8,              -- orientation de ce noeud par
                                               rapport au maître:
                                               0 = inconnue, 1 = identique,
                                               2 = inversée
  rsv1                  WORD1 (=0)          -- réservé, =0
  rsv2                  WORD1 (=0),         -- réservé, =0
  topo_counter          UNSIGNED6,          -- les six bits du compteur de
                                               topographie de ce noeud.
  individual_period     UNSIGNED8,          -- entier de 8 bits
                                               représentant la période
                                               attribuée par le maître,
                                               comme puissance de deux de
                                               la période de base, en ms.
  is_strong             ENUM8,              -- is_strong = 1: bus contrôlé
                                               par un maître fort,
                                               is_strong = 0: bus contrôlé
                                               par un maître faible
  number_of_nodes       UNSIGNED8 (0.0,63), -- nombre de noeuds dans la
                                               composition
  bottom_address        UNSIGNED8,          -- Node_Address du noeud avec
                                               l'adresse la plus basse
  top_address           UNSIGNED8,          -- Node_Address du noeud avec
                                               l'adresse la plus haute
  inaug_data_max_size  UNSIGNED8            -- copie de maxInaugDataSize
  nr_descriptors        UNSIGNED8           -- en conditions libres
                                               d'erreur, égal à
                                               number_of_nodes
```

```
  node_descriptions  ARRAY[nr_descriptors] OF
    {                                      -- liste de descriptions,
                                              commençant avec le noeud du
                                              bas, en ordre croissant
                                              d’adresse, consistant en:
    node_type           UNSIGNED8,         -- première partie de Node_Key
    node_version        UNSIGNED8,         -- deuxième partie de Node_Key
    sam                 BOOLEAN1           -- même direction que le maître
    rsv1                WORD1 (=0)         -- réservé, =0
    node_address        UNSIGNED6          -- adresse du noeud
    inauguration_data_size  UNSIGNED8      -- taille des données qui
                                              suivent
    inauguration_data  ARRAY [inauguration_data_size] OF
                        WORD8              -- données d’inauguration,
                                              structure fournie par
                                              l’utilisateur.
    }
  }
```

8.4.3.5 Write_Wtb_User_Report

8.4.3.5.1 Description

Définit les paramètres d’un nœud de Bus de Train.

8.4.3.5.2 Call_Write_Wtb_User_Report

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 25							
bus_id								ur7	ur6	ur5	ur4	ur3	ur2	ur1	ur0

```
Call_Write_Wtb_User_Report::= RECORD
  {
  bus_id              UNSIGNED8 (0..15),    -- indicatif de la liaison
  command             BITSET8
    {
    ur7       (0),                          -- Rapport utilisateur bit7
    ur6       (1),                          -- rapport utilisateur bit6
    ur5       (2),                          -- rapport utilisateur bit5
    ur4       (3),                          -- rapport utilisateur bit4
    ur3       (4),                          -- rapport utilisateur bit3
    ur2       (5),                          -- rapport utilisateur bit2
    ur1       (6),                          -- rapport utilisateur bit1
    ur0       (7),                          -- rapport utilisateur bit0
    }
  }
```

8.4.3.5.3 Reply_Write_Wtb_User_Report

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 25							
bus_id								reserved1							

```
Reply_Write_Wtb_User_Report::= RECORD
  {
  bus_id              UNSIGNED8 (0..15),  -- indicatif de la liaison
  reserved1           WORD8 (=0)          -- réservé
  }
```

8.4.3.6 Line_Fault_Location_Detection LFLD

8.4.3.6.1 Description

Le service recherche le défaut d'une ligne perturbée entre deux nœuds. Si cette Station ne comporte aucune connexion au bus de train, ce service renvoie une erreur et le Reply_Message se compose uniquement de l'en-tête.

Le service comporte plusieurs sous-commandes. Les sous-commandes suivantes sont fournies pour une application de gestion de réseau:

- Démarrer LFLD
- Extraire le résultat LFLD
- Annuler LFLD

Les sous-commandes suivantes sont fournies pour un usage interne.

- Indiquer le démarrage du LFLD au nœud final opposé
- Indiquer l'arrêt du LFLD au du nœud final opposé
- Démarrer le nœud de segmentation LFLD (nœud intermédiaire)
- Arrêter le nœud de segmentation LFLD (nœud intermédiaire)

Pour utiliser le service Line_Fault_Location_Detection, il convient d'appliquer un processus en plusieurs étapes. Il convient que l'application de diagnostic surveille les bits d'état de la ligne pour détecter toute perturbation. Si une perturbation de ligne permanante est détectée, il convient que l'application de diagnostic envoie la sous-commande 0 de Line_Fault_Location_Detection à un nœud d'extrémité pour démarrer le processus LFLD. Ensuite, l'application de diagnostic doit interroger le résultat LFLD avec la sous-commande 1 jusqu'à ce que le résultat indique sa disponibilité. Le résultat LFLD ne peut être extrait qu'une seule fois. Pour obtenir un nouveau résultat LFLD, le processus LFLD doit être relancé.

Pour annuler un processus LFLD en cours d'exécution, l'application de diagnostic doit émettre une sous-commande 2 Line_Fault_Location_Detection vers le nœud d'extrémité, sur lequel le processus LFLD a été démarré avec la sous-commande 0.

NOTE 1 En cas de plusieurs défauts sur la ligne perturbée, un seul peut être détecté.

NOTE 2 En présence de nœuds qui ne prennent pas en charge le service Line_Fault_Location_Detection, l'emplacement peut uniquement se trouver entre plusieurs nœuds.

NOTE 3 Si une inauguration se produit, le processus LFLD est annulé.

8.4.3.6.2 Call_Line_Fault_Location_Detection

0–7	8–15
tnm_key	sif_code = 26
bus_id	sub_command
par1	par2

```
Call_Line_Fault_Location_Detection::= RECORD
  {
  bus_id                UNSIGNED8 (0..15)   -- indicatif de la liaison
  sub_command           WORD8 (0..6)        -- sous-commande
  par1                  WORD8               -- paramètre 1 de la sous-
                                               commande LFLD
  par2                  WORD8               -- paramètre 2 de la sous-
                                               commande LFLD
  }
```

Les sous-commandes Line_Fault_Location_Detection suivantes sont définies:

- Sous-commande = 0: Démarrer le processus LFLD. Cette sous-commande peut uniquement être envoyée à un nœud d'extrémité.

par1 = 0, par2 = 0: Non utilisé.

- Sous-commande = 1: Extraire le résultat LFLD. Cette sous-commande peut uniquement être envoyée à un nœud d'extrémité.

par1 = 0, par2 = 0: Non utilisé.

- Sous-commande = 2: Annuler le processus LFLD. Cette sous-commande peut uniquement être envoyée à un nœud d'extrémité.

par1 = 0, par2 = 0: Non utilisé.

- Sous-commande = 3: Indiquer le début de LFLD au nœud d'extrémité opposé.

par1 = 0, par2 = 0: Non utilisé.

- Sous-commande = 4: Indiquer l'arrêt de LFLD au nœud d'extrémité opposé.

par1 = 0, par2 = 0: Non utilisé.

- Sous-commande = 5: Démarrer le nœud de segmentation LFLD (nœud intermédiaire).

par1 = *line* Ligne perturbée normalisée comme indiquée par le maître du bus du WTB (*line* = 0 indique la ligne A, *line* = 1 indique la ligne B)

par2 = *oe* adresse de nœud *oe* du nœud d'extrémité opposé

- Sous-commande = 6: Arrêter le nœud de segmentation LFLD (nœud intermédiaire).

par1 = *line* Ligne perturbée normalisée comme indiquée par le maître du bus du WTB (*line* = 0 indique la ligne A, *line* = 1 indique la ligne B)

par2 = 0 Non utilisé

8.4.3.6.3 Reply_Line_Fault_Location_Detection

0–7	8–15
tnm_key	sif_code = 26
bus_id	sub-command
result	ret1
ret2	reserved

```
Reply_Line_Fault_Location_Detection::= RECORD
  {
  bus_id                UNSIGNED8 (0..15)   -- indicatif de la liaison
  sub_command           WORD8 (0..6)        -- sous-commande
  result                WORD8               -- résultat d'exécution
  ret1                  WORD8               -- renvoie la valeur 1 de la
                                               sous-commande LFLD
  ret2                  WORD8               -- renvoie la valeur 2 de la
                                               sous-commande LFLD
  réservé               WORD8 (=0)          -- réservé
}
```

Les réponses suivantes à la sous-commande Line_Fault_Location_Detection sont définies:

- Sous-commande = 0 Démarrer le processus LFLD.

result = 0 Le processus LFLD a démarré

= 1 Le processus LFLD est déjà en cours d'exécution

= 4 commande envoyée au nœud intermédiaire: le processus LFLD n'a pas démarré

= 5 aucune ligne perturbée: le processus LFLD n'a pas démarré

= 6 les deux lignes (temporaires) sont perturbées: impossible de démarrer le processus LFLD

ret1, ret2 = 0 Non utilisé

- Sous-commande = 1 Extraire le résultat LFLD.

result = 0 Le processus LFLD n'a pas démarré, aucun résultat disponible

= 1 Le processus LFLD est en cours d'exécution, aucun résultat disponible

= 2 Le résultat LFLD est disponible

ret1 = *n1* une adresse de nœud *n1* délimitant le défaut de ligne

ret2 = *n2* autre adresse de nœud *n2* délimitant le défaut de ligne

- Sous-commande = 2 Annuler le processus LFLD.

result = 0 Le processus LFLD n'a pas démarré sur ce nœud d'extrémité

= 1 Le processus LFLD est annulé

ret1, ret2 = 0 Non utilisé

- Sous-commande = 3: Indiquer le démarrage de LFLD au nœud d'extrémité opposé.

result = 0: le processus LFLD a démarré au nœud d'extrémité opposé

ret1, ret2 = 0 Non utilisé.

- Sous-commande = 4: Indiquer l'arrêt de LFLD au nœud d'extrémité opposé.

result = 0: le processus LFLD s'est arrêté au nœud d'extrémité opposé ret1, ret2 = 0 Non utilisé.

- Sous-commande = 5: Démarrer le nœud de segmentation LFLD

result = 0: le nœud de segmentation LFLD a démarré

ret1, ret2 = 0 Non utilisé

- Sous-commande = 6: Arrêter le nœud de segmentation LFLD

result = 0 le nœud de segmentation LFLD est arrêté

ret1= *oe* adresse de nœud *oe* du nœud d'extrémité opposé, si une trame a été reçue

127, si aucune trame provenant du nœud d'extrémité opposé n'a été reçue

ret2 = 0 Relais de raccordement dans la direction 1 fermé

= 1 Relais de raccordement dans la direction 2 fermé

8.4.4 Services de Variables

8.4.4.1 Read_Ports_Configuration

8.4.4.1.1 Description

Extrait la liste des ports configurés avec leur F_code respectifs (qui contient la taille) et leurs attributs.

8.4.4.1.2 Call_Read_Ports_Configuration

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="8">tnm_key</td><td colspan="8">sif_code = 30</td></tr>
<tr><td colspan="8">bus_id</td><td colspan="8">reserved1 = 0</td></tr>
</table>

```
Call_Read_Ports_Configuration::= RECORD
  {
  bus_id              UNSIGNED8,         -- (0..15)
  reserved1           WORD8 (=0)         -- réservé
  }
```

8.4.4.1.3 Reply_Read_Ports_Configuration

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="8">tnm_key</td><td colspan="8">sif_code = 30</td></tr>
<tr><td colspan="16">nr_ports</td></tr>
<tr><td colspan="16">ports_list: ARRAY [nr_ports] OF</td></tr>
<tr><td colspan="4">bus_id</td><td colspan="12">port_address</td></tr>
<tr><td colspan="4">f_code</td><td>src</td><td>snk</td><td>twc</td><td>frc</td><td colspan="8">port_size</td></tr>
</table>

```
Reply_Read_Ports_Configuration::= RECORD
  {
  nr_ports            UNSIGNED16,        -- nombre de ports dans ce
                                            Traffic_Store (jusqu'à 4096)
  ports_list          ARRAY [nr_ports] OF
    {
```

```
    bus_id              WORD4,               -- traffic store
    port_address        UNSIGNED12,          -- adresse du port
    f_code              F_Code,              -- F_code du port
    port_config         BITSET4
      {
      src                                    -- port de l'éditeur (émetteur)
      snk                                    -- port du souscripteur
                                                (destinataire)
      twc                                    -- transfert avec somme de
                                                contrôle
      frc                                    -- port forcé
      },
    port_size           UNSIGNED8            -- taille maximale du port en
                                                octets (seulement pour WTB)
    }
  }
```

8.4.4.2 Write_Ports_Configuration

8.4.4.2.1 Description

Active ou désactive les ports.

8.4.4.2.2 Call_Write_Ports_Configuration

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 31							
nr_ports															
ports_list: ARRAY [nr_ports] OF															
bus_id				port address											
f_code				src	snk	twc	frc	port_size							

```
Call_Write_Ports_Configuration::= RECORD
  {
  nr_ports              UNSIGNED16,          -- nombre de ports dans ce
                                                Traffic_Store (jusqu'à 4096)
  ports_list            ARRAY [nr_ports] OF
    {
    bus_id              UNSIGNED8 (0..15),   -- indicatif du traffic store
                                                vu par la station
    port_address        UNSIGNED12,          -- adresse du port
    f_code              F_Code,              -- F_code (voir objet de
                                                configuration de port)
    port_config         BITSET4
      {
      src                                    -- active le port de l'éditeur
                                                (émetteur)
      snk                                    -- active le port du
                                                souscripteur (destinataire)
      twc                                    -- active le transfert avec
                                                somme de contrôle
      frc                                    -- force le port à sa valeur de
                                                défaut
      },
    port_size           UNSIGNED8            -- taille maximale du port en
                                                octets (seulement pour WTB)
    }
  }
```

8.4.4.2.3 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)   -- indicatif du traffic store
                                              vu par la station
  reserved1            WORD8 (=0)          -- réservé
  }
```

8.4.4.3 Read_Variables

8.4.4.3.1 Description

Lit la valeur, la variable de contrôle et le niveau de rafraîchissement d'une grappe de variables.

Chaque variable est identifiée par sa position, celle de sa variable de contrôle à l'intérieur de son Traffic_Store, son type et sa taille.

Les variables peuvent être analysées par PV_Sets ou par PV_Clusters.

Si plusieurs variables consécutives appartiennent au même dataset, leur accès doit être cohérent (par PV_Set).

L'Agent doit répondre par une liste de valeurs dans le même ordre que les variables énumérées dans le Call_Message.

Les valeurs des variables doivent être transmises dans les messages de gestion selon le format qui aurait été le leur lors de leur transmission comme variables de processus sur le WTB (en big-endian).

Chaque variable doit commencer sur une nouvelle limite de mot.

Les valeurs qui ne remplissent pas un mot de 16 bits doivent être justifiées à droite et, si elles sont signées, étendues par leur signe sur leur gauche.

Les variables supérieures à 16 bits, mais dont la taille n'est pas un multiple de 16 bits, doivent être justifiées à droite et complétées par des "0" jusqu'à la prochaine limite de 16 bits.

EXEMPLE 1 Un entier de 8 bits avec la valeur '0111 1111'B (+127) est transmis comme '0000 0000 0111 1111'B.

EXEMPLE 2 Un entier de 8 bits avec la valeur '1111 1111'B (-1) est transmis comme '1111 1111 1111 1111'B.

8.4.4.3.2 Call_Read_Variables

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="8">tnm_key</td><td colspan="8">sif_code = 32</td></tr>
<tr><td colspan="16">nr_vars</td></tr>
<tr><td colspan="16">variables_list: ARRAY [nr_vars] OF</td></tr>
<tr><td colspan="4">bus_id</td><td colspan="12">port_address</td></tr>
<tr><td colspan="8">var_size</td><td colspan="8">var_type</td></tr>
<tr><td colspan="16">var_offset</td></tr>
<tr><td colspan="16">chk_offset</td></tr>
</table>

```
Call_Read_Variables::= RECORD
  {
  nr_vars               UNSIGNED16,          -- nombre de variables dans la
                                                liste.
  variables_list        ARRAY [ nr_vars ] OF
    {                                        -- liste avec une entrée pour
                                                chaque variable.
    bus_id              UNSIGNED4,           -- indicatif du Traffic_store
    port_address        WORD12,              -- adresse de port dans le
                                                Traffic_store
    var_size            UNSIGNED8,           -- (types simples: Taille en
                                                bits ou types structurés:
                                                nombre d’éléments selon 6.2
                                                (RTP)
    var_type            UNSIGNED8,           -- type de la variable
                                                selon 6.2 (RTP)
    var_offset          UNSIGNED16,          -- décalage de la variable
    chk_offset          UNSIGNED16           -- décalage de la variable de
                                                contrôle (sinon ‘FFFF’H).
    }
  }
```

8.4.4.3.3 Reply_Read_Variables

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="8">tnm_key</td><td colspan="8">sif_code = 32</td></tr>
<tr><td colspan="16">nr_vars</td></tr>
<tr><td colspan="16">values_list: ARRAY [nr_vars] OF:</td></tr>
<tr><td colspan="16">value (justifiée à droite si inférieure à 16 bits)</td></tr>
<tr><td colspan="2">check_var</td><td colspan="14">freshness counter</td></tr>
</table>

```
Reply_Read_Variables::=RECORD
  {
  nr_vars               UNSIGNED16,          -- nombre de variables dans la
                                                liste.
  values_list           ARRAY [ nr_vars ] OF
    {                                        -- pour chaque variable de la
                                                liste et dans le même ordre:
    value               ANY,                 -- valeur avec un format donné
                                                par Size et Type du PV_NAME,
                                                aligné sur une limite de mot
                                                de 16 bits.
```

```
                                          CHARACTER8 est une
                                          exception, car il est
                                          interprété comme un cas
                                          particulier de ARRAY [0..0]
                                          OF CHARACTER8: Le caractère
                                          occupe l'octet d'ordre le
                                          plus élevé, alors que
                                          l'octet d'ordre le plus bas
                                          contient '00'h
    check_var          ANTIVALENT2,    -- variable de contrôle
                                          associée, ou sinon '11'B.
    freshness          UNSIGNED14      -- valeur du niveau de
                                          rafraîchissement de la
                                          variable en millisecondes
                                          (maximum: 4096 ms).
    }
  }
```

NOTE Une Check_Variable peut également être transportée par une autre variable de type ANTIVALENT2.

8.4.4.4 Write_Force_Variables

8.4.4.4.1 Description

Force une grappe de variables à une valeur donnée.

Le forçage d'une variable doit activer le bit 'frc' dans le Device_Status du Traffic_Store du MVB correspondant et dans le Station_Status.

La valeur spécifiée par la variable de contrôle doit être attribuée au champ de contrôle.

Chaque variable doit commencer sur une nouvelle limite de mot.

Les valeurs qui ne remplissent pas un mot de 16 bits doivent être justifiées à droite et, si elles sont signées, étendues par leur signe sur leur gauche.

Les variables supérieures à 16 bits, mais dont la taille n'est pas un multiple de 16 bits, doivent être justifiées à droite et complétées par des "0" jusqu'à la prochaine limite de 16 bits.

NOTE L'état du Bus de Train ne contient pas de bit 'frc', mais cette information peut être déduite du Station_Status.

8.4.4.4.2 Call_Write_Force_Variables

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 33							
nr_vars															
variables_list: ARRAY [nr_vars] OF															
bus_id				port_address											
var_size								var_type							
var_offset															
chk_offset															
values_list: ARRAY ALIGN16 [nr_vars] OF															
forced_value															

```
Call_Write_Force_Variables::= RECORD
  {
  nr_vars                UNSIGNED16,         -- nombre de variables dans la
                                                liste
  variables_list         ARRAY [ nr_vars ] OF
                                             -- Une entrée pour chaque
                                                variable
    {
    bus_id               UNSIGNED4,          -- indicatif du traffic_store
    port_address         WORD12,             -- adresse de port dans le
                                                traffic_store
var_size                 UNSIGNED8,          -- taille en bits (types
                                                simples) ou éléments (types
                                                structurés) selon 6.2 (RTP)
    var_type             UNSIGNED8,          -- type de la variable
                                                selon 6.2 (RTP)
    var_offset           UNSIGNED16,         -- décalage de la variable
    chk_offset           UNSIGNED16          -- décalage de la variable de
                                                contrôle (sinon ‘FFFF’H).
},
  values_list            ARRAY ALIGN16 [nr_vars] OF
    {                                        -- liste de valeurs à forcer,
                                                dans le même ordre que la
                                                variables_list (les mêmes
                                                règles que pour les
                                                variables de lecture
                                                s’appliquant)
  forced_value           ANY                 -- valeur de la variable au
                                                format donné par PV_Name,
                                                telle qu’elle serait
                                                transmise sur le bus.
    }
  }
```

8.4.4.4.3 Reply_Write_Force_Variables

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 33							

```
Reply_Write_Force_Variables::= RECORD
  {}                                         -- aucun paramètre
```

8.4.4.5 Write_Unforce_Variables

8.4.4.5.1 Description

Lève le forçage des variables énumérées. Si ce service aboutit, il doit réinitialiser le bit "frc" de l'état de la couche de liaison correspondante, ainsi que le bit 'frc' du Station_Status si le forçage de toutes les variables de la station a été levé.

8.4.4.5.2 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,          -- nombre de variables dans la
                                               liste.
  variables_list        ARRAY [ nr_vars ] OF
                                            -- Une entrée pour chaque
                                               variable non forcée
    {
    bus_id              UNSIGNED4,          -- indicatif du traffic_store
    port_address        WORD12,             -- adresse de port dans le
                                               traffic_store
var_size                UNSIGNED8,          -- taille en bits (types
                                               simples) ou nombre
                                               d'éléments (types
                                               structurés) selon 6.2 (RTP)
    var_type            UNSIGNED8,          -- type de la variable
                                               selon 6.2 (RTP)
    var_offset          UNSIGNED16,         -- décalage de la variable
    chk_offset          UNSIGNED16          -- décalage de la variable de
                                               contrôle (sinon 'FFFF'H).
}
  }
```

8.4.4.5.3 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
  {}                                        -- aucun paramètre
```

8.4.4.6 Write_Unforce_All

8.4.4.6.1 Description

Lève le forçage de toutes les variables d'un Traffic_Store particulier.

Si ce service aboutit, il désactive le bit "frc" du Device_Status correspondant à celui du Traffic_Store et dans le Station_Status, une fois le forçage de tous les Traffic_Stores levé.

8.4.4.6.2 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 indiquant les
                                                Traffic_Stores à libérer,
                                                Traffic_Store 0 étant
                                                représenté par le bit de
                                                décalage 0.
  }
```

8.4.4.6.3 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
  {}                                         -- aucun paramètre
```

8.4.4.7 Read_Variable_Bindings

8.4.4.7.1 Description

Lit la liaison de toutes les variables prises en charge par une station. L'ordre est aléatoire.

NOTE Ce service prend en charge le triage des variables internes du dispositif avec les variables de réseau.

8.4.4.7.2 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
  {
  }
```

8.4.4.7.3 Reply_Read_Variable_Bindings

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="8">tnm_key</td><td colspan="8">sif_code = 38</td></tr>
<tr><td colspan="16">nr_vars</td></tr>
<tr><td colspan="16">variables_list: ARRAY[nr_vars] OF</td></tr>
<tr><td colspan="16">variable_name: STRING32</td></tr>
<tr><td colspan="8">(CHARACTER8)</td><td colspan="8">CHARACTER8 or ‘00’H</td></tr>
<tr><td colspan="8">var_properties</td><td colspan="8">individual_period</td></tr>
<tr><td colspan="16">standard_type</td></tr>
<tr><td colspan="4">bus_id</td><td colspan="12">port_address</td></tr>
<tr><td colspan="8">var_ size</td><td colspan="8">var_type</td></tr>
<tr><td colspan="16">var_offset</td></tr>
<tr><td colspan="16">chk_offset</td></tr>
</table>

```
Reply_Read_Variable_Bindings::= RECORD
  {
  nr_vars              UNSIGNED16,          -- nombre de variables dans la
                                               liste.
  variables_list       ARRAY [ nr_vars ] OF
                                            -- une entrée pour chaque
                                               variable décrite
    {
    variable_name      STRING32,            -- nom local de la variable
    var_properties     BITSET8
      {
      bnd    (0)                            -- 1 variable à lier,
                                               0 aucune action
      phl    (1),                           -- 1 adresse physique
                                               (mémoire),
                                               0 adresse logique (port)
      reg    (6),                           -- 1 variable régulière,
                                               0 variable de maintenance
      imp    (7)                            -- 1 variable importée,
                                               0 variable exportée
      },
    individual_period  UNSIGNED8            -- période individuelle  de la
                                               variable comme puissance
                                               de 2  de 1 ms
                                               (4 = 16 ms, par exemple).
    standard_type      ENUM16,              -- type standard défini par
                                               l'application
    bus_id             UNSIGNED4,           -- traffic_store auquel la
                                               variable est liée
    port_address       WORD12,              -- adresse de port auquel la
                                               variable est liée
    var_size           UNSIGNED8,           -- taille en bits (types
                                               simples) ou nombre
                                               d'éléments (types
                                               structurés) selon 6.2 (RTP)
    var_type           UNSIGNED8,           -- code du type de la variable
                                               selon 6.2 (RTP)
    var_offset         UNSIGNED16,          -- décalage de la variable dans
                                               le dataset
    chk_offset         UNSIGNED16           -- décalage de la variable de
                                               contrôle (sinon ‘FFFF’H).
    }
  }
```

8.4.4.8 Write_Variable_Bindings

8.4.4.8.1 Description

Lie ou délie un certain nombre de variables à un traffic_store et une adresse de port.

NOTE Ce service prend en charge la configuration ROSIN.

8.4.4.8.2 Call_Write_Variable_Bindings

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 39							
nr_vars															
variables_list: ARRAY[nr_vars] OF															
variable_name: STRING32															
								CHARACTER8 ou ‘00’H							
var_properties								standard_type							
individual_period															
bus_id				port_address											
var_size								var_type							
var_offset															
chk_offset															

```
Call_Write_Variable_Bindings::= RECORD
  {
  nr_vars              UNSIGNED16,         -- nombre de variables à lier.
  variables_list       ARRAY [ nr_vars ] OF
    {                                      -- tableau avec une entrée pour
                                              chaque variable à lier
    variable_name      STRING32            -- nom local de la variable à
                                              lier ou délier
    var_properties     BITSET8
      {
      bnd    (0)                           -- 1 variable à lier,
                                              0 aucune action
      phl    (1),                          -- 1 adresse physique
                                              (mémoire),
                                              0 adresse logique (port)
      reg    (6),                          -- 1 variable régulière,
                                              0 variable de maintenance
      imp    (7)                           -- 1 variable importée,
                                              0 variable exportée
      },
    standard_type      ENUM8,              -- type standard défini par
                                              l'application
    individual_period  UNSIGNED16          -- période individuelle de la
                                              variable en millisecondes.
    standard_type      ENUM8,              -- code d'application de type
    bus_id             UNSIGNED4,          -- traffic_store avec lequel la
                                              variable est liée ou déliée.
    port_address       WORD12,             -- port avec lequel la variable
                                              est liée (déliée si le port
                                              est 0)
    var_size           UNSIGNED8,          -- taille en bits (types
                                              simples) ou nombre
                                              d'éléments (types
                                              structurés) selon 6.2.
    var_type           UNSIGNED8,          -- type selon 6.2 (RTP)
    var_offset         UNSIGNED16,         -- décalage de la variable
    chk_offset         UNSIGNED16,         -- décalage de la variable de
                                              contrôle (’FFFF’h si non
                                              utilisée).
```

```
    }
  }
```

8.4.4.8.3 Reply_Write_Variable_Bindings

0–7	8–15
tnm_key	sif_code = 39

```
Reply_Write_Variable_Bindings::= RECORD
  {
  }
```

8.4.4.9 Write_Attach_Port

8.4.4.9.1 Description

Attache les ports du Traffic_Store à des entrées et sorties spécifiques (stations de Classe 2).

8.4.4.9.2 Call_Write_Attach_Port

0–3	4–7	8–15
tnm_key		sif_code = 29
nr_ports		
port_point_list: ARRAY[nr_vars] OF		
bus_id	port_address	
point		
filter		
gain		
offset		

```
Call_Write_Attach_Port::= RECORD
  {
  nr_ports              UNSIGNED16,         -- nombre de ports considérés
  port_point_list       ARRAY [nr_ports] OF
    {                                       -- liste des ports et des
                                               points de chaque élément:
    ds_name             RECORD
      {
      bus_id            UNSIGNED4 (0..15),  -- bus_id
                                               (Traffic_Store) = (0 par
                                               défaut)
      port_address      WORD12              -- adresse de port de 12 bits
                                               (doit être divisible par 2)
      }
    point_descriptor    RECORD              -- descripteur du point,
                                               contenant:
      {
      point             UNSIGNED16,         -- indicatif à 16 bits du point
                                               d'entrée/sortie
      filter            UNSIGNED16,         -- constante de temps du
                                               filtre, en multiples de
                                               10 ms (ou 0 si non utilisé)
      gain              UNSIGNED16,         -- valeur du gain analogique
      offset            INTEGER16           -- valeur du décalage d'une
                                               valeur analogique
      }
    }
  }
```

8.4.4.9.3 Reply_Write_Attach_Port

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 39							

```
Reply_Write_Attach_Port::= RECORD
  {}                                        -- aucun paramètre
```

8.4.5 Services de messagerie

8.4.5.1 Read_Messenger_Status

8.4.5.1.1 Description

Extrait l'état du messager et ses compteurs de statistiques.

8.4.5.1.2 Call_Read_Messenger_Status

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 40							

```
Call_Read_Messenger_Status::= RECORD
  {}                                        -- aucun paramètre
```

8.4.5.1.3 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),        -- réservé
  messenger_name       STRING32,           -- version du logiciel du
                                              messager, de préférence au
                                              format:
                                              xxxx-Vz.z-dd.mm.yy
  send_time_out        UNSIGNED8,          -- temporisation de réémission
                                              du producteur, en multiples
                                              de 64 ms
  alive_time_out       UNSIGNED8,          -- temporisation de déconnexion
                                              du consommateur, en secondes
  ack_time_out         UNSIGNED8,          -- temporisation après laquelle
                                              le répondeur accuse
                                              réception de tous les
                                              paquets de données reçus, en
                                              multiple de 64 ms
  credit               UNSIGNED8,          -- nombre de paquets de données
                                              que le producteur peut
                                              envoyer avant de recevoir un
                                              accusé de réception pour
                                              l'un d'eux
  reserved2            WORD8 (=0),         -- réservé
  packet_size          UNSIGNED8,          -- taille d'un paquet en octets
  instances            UNSIGNED8,          -- nombre d'instances prises en
                                              charge pour chaque
                                              répondeur.
  multicast_window     UNSIGNED8,          -- taille de fenêtre du
                                              protocole de distribution.
                                              Si 0, la diffusion n'est pas
                                              prise en charge.
  messages_sent        UNSIGNED32,         -- compteur (avec bouclage)
                                              comptant le nombre de
                                              messages envoyés par cette
                                              station
  messages_received    UNSIGNED32,         -- compteur (avec bouclage)
                                              comptant le nombre de
                                              messages reçus par cette
                                              station
  packets_sent         UNSIGNED32,         -- compteur (avec bouclage)
                                              comptant le nombre de
                                              paquets envoyés sur cette
                                              station
  packet_retries       UNSIGNED32,         -- compteur (avec bouclage)
                                              comptant le nombre de
                                              paquets répétés par le
                                              protocole point à point
  multicast_retries    UNSIGNED32          -- compteur (avec bouclage)
                                              comptant le nombre de
                                              paquets répétés par le
                                              protocole de distribution
  }
```

8.4.5.2 Write_Messenger_Control

8.4.5.2.1 Description

Définit les paramètres du messager.

8.4.5.2.2 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	pkr	pks	mgr	mgs	multicast_window							

```
Call_Write_Messenger_Control::= RECORD
  {
  reserved1             WORD16 (=0),         -- réservé
  messenger_name        STRING32,            -- version du logiciel du
                                                messager, de préférence au
                                                format:
                                                xxxx-Vz.z-dd.mm.yy
  send_time_out         UNSIGNED8,           -- temporisation de réémission
                                                du producteur, en multiples
                                                de 64 ms
  alive_time_out        UNSIGNED8,           -- temporisation de déconnexion
                                                du consommateur, en secondes
  ack_time_out          UNSIGNED8,           -- temporisation après laquelle
                                                le répondeur accuse
                                                réception de tous les
                                                paquets reçus, en multiple
                                                de 64 ms
  credit                UNSIGNED8,           -- nombre de paquets de données
                                                qui peuvent être envoyés
                                                avant de recevoir un accusé
                                                de réception
  reserved2             WORD8 (=0),          -- réservé
  packet_size           UNSIGNED8,           -- taille d'un paquet, en
                                                octets
  clear_counter         BITSET8              -- remet à zéro les compteurs
                                                suivants:
    {
    rsv1,                                    -- réservé
    rsv2,                                    -- réservé
    rsv3,                                    -- réservé
    mcr,                                     -- compteur de répétition du
                                                protocole de distribution
    pkr,                                     -- compteur de répétitions de
                                                paquet
    pks,                                     -- compteur de paquets envoyés
    mgr,                                     -- compteur de messages reçus
    mgs                                      -- compteur de messages envoyés
    },
  multicast_window      UNSIGNED8            -- taille de fenêtre du
                                                protocole de distribution.
  }                                          -- Si 0, la diffusion n'est pas
                                                prise en charge.
```

8.4.5.2.3 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
  {}                                         -- aucun paramètre
```

8.4.5.3 Read_Function_Directory

8.4.5.3.1 Description

Lit le Function_Directory d'une station.

8.4.5.3.2 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
  {}                                         -- aucun paramètre
```

8.4.5.3.3 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)            -- réservé
  nr_functions        UNSIGNED8,            -- nombre d'entrées dans la
                                               liste

  function_list       ARRAY[nr_functions]  OF
    {                                       -- tableau de paires
                                               Function/Station, comportant
                                               chacune:
    function_id       UNSIGNED8,            -- Function_Identifier
    station_id        UNSIGNED8             -- Station_Identifier
                                               correspondant
    }
  }
```

8.4.5.4 Write_Function_Directory

8.4.5.4.1 Description

Écrit le Function_Directory d'une station.

8.4.5.4.2 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),                        -- n'efface pas, mais substitue
                                               les éléments
    CLEARFIRST  (1)                         -- efface le répertoire avant
                                               d'écrire
    },
  nr_functions          UNSIGNED8,          -- nombre d'entrées dans la
                                               liste
  function_list         ARRAY[nr_functions]  OF
    {                                       -- tableau de paires
                                               Function/Station, comportant
                                               chacune:
    function_id         UNSIGNED8,          -- Function_Identifier
    station_id          UNSIGNED8           -- Station_Identifier
                                               correspondant
    }
  }
```

8.4.5.4.3 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
  {}                                        -- aucun paramètre
```

8.4.5.5 Read_Station_Directory

8.4.5.5.1 Description

Lit le Station_Directory d'une station (s'il existe).

8.4.5.5.2 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
  {}                                        -- aucun paramètre
```

8.4.5.5.3 Reply_Read_Station_Directory

0–7	8–15
tnm_key	sif_code = 44
reserved1	nr_stations
station_list: ARRAY [nr_stations] OF	
station_id	next_station_id
bus_id	reserved2
device_address	

```
Reply_Read_Station_Directory::= RECORD
  {
  reserved1              WORD8 (=0),           -- réservé
  nr_stations            UNSIGNED8,            -- nombre de stations dans la
                                                  liste
  station_list           ARRAY [nr_stations] OF
    {                                          -- liste de stations avec la
                                                  Next_Station correspondante,
                                                  comprenant:
    station_id           UNSIGNED8,            -- Station_Identifier
    next_station_id      UNSIGNED8,            -- Station_Identifier de
                                                  Next_Station
    bus_id               UNSIGNED8 (0..15),    -- indicatif de la liaison sur
                                                  laquelle se trouve
                                                  Next_Station
    reserved2            WORD8 (=0),           -- réservé
    device_address       UNSIGNED16            -- adresse du dispositif qui
                                                  porte Next_Station
    }
  }
```

8.4.5.6 Write_Station_Directory

8.4.5.6.1 Description

Écrit le Station_Directory d'une station (s'il existe).

8.4.5.6.2 Call_Write_Station_Directory

0–7	8–15
tnm_key	sif_code = 45
clear_directory	nr_stations
station_list: ARRAY[nr_stations] OF	
station_id	next_station_id
bus_id	reserved1
device_address	

```
Call_Write_Station_Directory::= RECORD
  {
  clear_directory        ENUM8
    {
    REPLACE     (0),                           -- n'efface pas, mais substitue
                                                  les éléments
    CLEARFIRST  (1)                            -- efface le répertoire avant
                                                  d'écrire
```

```
    },
  nr_stations          UNSIGNED8,           -- nombre de stations dans la
                                               liste
  station_list         ARRAY [nr_stations] OF
    {                                       -- liste de stations avec la
                                               Next_Station correspondante,
                                               comprenant:
    station_id         UNSIGNED8,           -- Station_Identifier
    next_station_id    UNSIGNED8,           -- Station_Identifier de
                                               Next_Station
    bus_id             UNSIGNED8 (0..15),   -- indicatif de la liaison sur
                                               laquelle se trouve
                                               Next_Station
    reserved1          WORD8 (=0),          -- réservé
    device_address     UNSIGNED16           -- adresse du dispositif qui
                                               porte Next_Station
    }
  }
```

8.4.5.6.3 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
  {}                                        -- aucun paramètre
```

8.4.5.7 Read_Group_Directory

8.4.5.7.1 Description

Lit le Group_Directory d'une station (s'il existe).

8.4.5.7.2 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
  {}                                        -- aucun paramètre
```

8.4.5.7.3 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							
g7	g6	g5	g4	g3	g2	g1	g0	g15	g14	g13	g12	g11	g10	g9	g8
g23															g24
g39															g40
g55								g63							g56

```
Reply_Read_Group_Directory::= RECORD
  {
  group_list           BITSET64             -- un bit activé pour chacun
                                               des 64 groupes possibles
                                               auquel une station peut
```

```
                                               appartenir, le bit du groupe
                                               0 ayant le décalage 0.
  }
```

8.4.5.8 Write_Group_Directory

8.4.5.8.1 Description

Écrit le Group_Directory d'une station (s'il existe).

8.4.5.8.2 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							
g7	g6	g5	g4	g3	g2	g1	g0	g15	g14	g13	g12	g11	g10	g9	g8
g23															g24
g39															g40
g55								g63							g56

```
Call_Write_Group_Directory::= RECORD
  {
  group_list           BITSET64           -- un bit activé pour chacun
                                             des 64 groupes auquel une
                                             station peut appartenir, le
                                             bit du groupe 0 ayant le
                                             décalage 0.
}
```

8.4.5.8.3 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
  {}                                      -- aucun paramètre
```

8.4.5.9 Read_Node_Directory

8.4.5.9.1 Description

Lit le Node_Directory d'un nœud (s'il existe).

8.4.5.9.2 Call_Read_Node_Directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 48							

```
Call_Read_Node_Directory::= RECORD
  {}                                      -- aucun paramètre
```

8.4.5.9.3 Reply_Read_Node_Directory

0–7	8–15
tnm_key	sif_code = 48
reserved1	nr_nodes
nodes_list: ARRAY [nr_nodes] OF	
node_address	reserved2
device_address	

```
Reply_Read_Node_Directory::= RECORD
  {
  reserved1             WORD8 (=0),          -- réservé
  nr_nodes              UNSIGNED8,           -- nombre de noeuds dans la
                                                liste
  nodes_list            ARRAY [nr_nodes] OF
    {                                        -- liste des noeuds avec la
                                                Device_Address
                                                correspondante, comprenant:
    node_address        UNSIGNED8,           -- Node_Address à 8 bits
    reserved2           WORD8 (=0),          -- réservé
    device_address      UNSIGNED16           -- Device_Address des noeuds
    }
  }
```

8.4.5.10 Write_Node_Directory

8.4.5.10.1 Description

Ecrit le Node_Directory d’un nœud (s’il existe).

8.4.5.10.2 Call_Write_Node_Directory

0–7	8–15
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),                        -- n’efface pas, mais substitue
                                                les éléments
    CLEARFIRST  (1)                          -- efface le répertoire avant
                                                d’écrire
    },
  nr_nodes              UNSIGNED8,           -- nombre de noeuds dans la
                                                liste
  nodes_list            ARRAY [nr_nodes] OF
    {                                        -- liste des noeuds avec la
                                                Device_Address
                                                correspondante, comprenant:
    node_address        UNSIGNED8,           -- Node_Address à 8 bits
    reserved1           WORD8 (=0),          -- réservé
    device_address      UNSIGNED16           -- Device_Address des noeuds
    }
  }
```

8.4.5.10.3 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
  {}                                            -- aucun paramètre
```

8.4.6 Services de domaine

8.4.6.1 Read_Memory

8.4.6.1.1 Description

Ce service lit en une seule opération plusieurs zones de mémoire, chacune étant constituée d'une suite d'éléments consécutifs identiques, dont la taille est un multiple entier d'un octet, chaque octet ayant une adresse de mémoire.

L'alignement doit être respecté de la manière suivante:

- si la taille d'un élément est de 1 octet, la zone peut commencer à une adresse paire ou impaire;
- si la taille d'un élément est de 2 octets, la zone de mémoire doit commencer à une adresse paire;
- si la taille d'un élément est de 4 octets, la zone doit commencer à une adresse divisible par 4.

8.4.6.1.2 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,           -- nombre de zones
                                               individuelles à lire
  region_list          ARRAY[nr_regions] OF
    {                                       -- liste des zones, chacune
                                               comprenant:
    base_address       UNSIGNED32,          -- adresse de base de la zone
    nr_items           UNSIGNED16,          -- taille de la zone, en
                                               multiples de la taille de
                                               l'élément
    reserved2          WORD8 (=0),          -- réservé
    item_size          UNSIGNED8            -- taille de chaque élément en
                                               octets, valeurs permises: 1,
                                               2, 4.
    }
  }
```

8.4.6.1.3 Reply_Read_Memory

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td colspan="8">tnm_key</td><td colspan="8">sif_code = 50</td></tr>
<tr><td colspan="8">reserved1</td><td colspan="8">nr_regions</td></tr>
<tr><td colspan="16">region_values: ARRAY[nr_regions] OF</td></tr>
<tr><td colspan="16">nr_octets</td></tr>
<tr><td colspan="16">item_value_list: ARRAY ALIGN16 [nr_octets] OF WORD8</td></tr>
<tr><td colspan="8">premier octet à adresse paire
(ou '00'H si le début est impair)</td><td colspan="8">deuxième octet si le début est impair</td></tr>
<tr><td colspan="16">item_value</td></tr>
<tr><td colspan="8">dernier octet (si le début est impair et la taille de la zone est paire, ou le début est pair et la taille de la zone impaire)</td><td colspan="8">dernier octet (ou '00'H si le début est impair et la taille de la zone est paire, ou si le début est pair et la taille de la zone impaire)</td></tr>
</table>

```
Reply_Read_Memory::=    RECORD
```

```
{
reserved1           WORD8 (=0),           -- réservé
nr_regions          UNSIGNED8,            -- nombre de zones
                                             individuelles à lire
region_values       ARRAY[nr_regions] OF
  {                                       -- tableau des region_values,
                                             comprenant:
  nr_octets         UNSIGNED16,           -- le nombre d'octets dans le
                                             champ transporté, y compris
                                             les octets de remplissage.
  item_value_list   ARRAY ALIGN16 [nr_octets] OF
    {                                     -- tableau de [nr_octets]
                                             valeurs, comprenant pour
                                             chacune:
    item_value      WORD8                 -- valeurs de l'élément,
                                             transmises dans l'ordre
                                             continu.
    }                                     -- le premier octet doit être
                                             un octet de remplissage si
                                             la
  }                                       -- zone commence par une
                                             adresse impaire.
}
```

NOTE Les octets situés à une adresse de mémoire paire sont toujours transmis avec un décalage pair dans le message. Si la zone de mémoire commence à une adresse impaire, le premier élément transmis n'a aucune signification. A l'inverse, si la zone de mémoire commence à une adresse paire et que le nombre d'octets est impair, le dernier octet transmis n'a aucune signification. Si l'adresse de base est impaire et que le nombre d'octets est pair, le premier et le dernier octet n'ont aucune signification. Le champ 'nr_octets' contient le nombre effectif d'octets transmis et n'est donc égal à la taille de la zone que si cette dernière commence à une adresse paire et que le nombre d'octets est pair.

8.4.6.2 Write_Memory

8.4.6.2.1 Description

Ce service écrit en une seule opération plusieurs zones de mémoire, chacune comportant un nombre identique d'éléments d'une taille donnée.

8.4.6.2.2 Call_Write_Memory

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
tnm_key	sif_code = 51
reserved1	nr_regions
region_list: ARRAY [nr_regions] OF	
base address	
nr_items	
reserved2	item_size
region_value_list: ARRAY [nr_regions] OF	
item_value_list: ARRAY [nr_items] OF	
premier octet si nr_octets est pair, sinon sans signification	premier octet si nr_octets impair
octets pairs	octets impairs
dernier octet (si nr_octets est impair)	dernier octet (ou '00'H si nr_octets est impair)

```
Call_Write_Memory::= RECORD
  {
  reserved1            WORD8 (=0),          -- réservé
  nr_regions           UNSIGNED8,           -- nombre de zones à écrire
  region_list          ARRAY[nr_regions) OF
    {                                       -- liste des zones, chacune
                                               comprenant:
    base_address       UNSIGNED32,          -- l'adresse de base de la zone
                                               (peut être impaire)
    nr_items           UNSIGNED16,          -- taille de la zone, en
                                               multiples de la taille de
                                               l'élément
    reserved2          WORD8 (=0),          -- réservé
    item_size          UNSIGNED8            -- taille de chaque élément en
                                               octets, valeurs permises: 1,
                                               2, 4.
    },
  region_value_list    ARRAY[nr_regions] OF
    {                                       -- tableau des region_values,
                                               comprenant:
    item_value_list    ARRAY ALIGN16 [nr_items] OF
      {                                     -- tableau de 'nr_items'
                                               valeurs, chacune comprenant:
      ONE_OF [item_size]                    -- transmise dans l'ordre
                                               continu.
        {
        1:             WORD8,               -- les octets aux adresses
                                               paires sont transmis en
                                               premier
        2:             WORD16,              -- écrire doublet par doublet
        4:             WORD32               -- écrire quadlet par quadlet
        }
      }
    }
  }
```

NOTE Les octets situés à une adresse de mémoire paire sont toujours transmis avec un décalage pair dans le message. Si la zone de mémoire commence à une adresse impaire, le premier élément transmis n'a aucune signification. A l'inverse, si la zone de mémoire commence à une adresse paire et que le nombre d'octets est impair, le dernier octet transmis n'a aucune signification. Si l'adresse de base est impaire et que le nombre d'octets est pair, le premier et le dernier octet n'ont aucune signification. Le champ 'nr_octets' contient le nombre effectif d'octets transmis et n'est donc égal à la taille de la zone que si cette dernière commence à une adresse paire et que le nombre d'octets est pair.

8.4.6.2.3 Reply_Write_Memory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 51							

```
Reply_Write_Memory::= RECORD
  {}                                        -- aucun paramètre
```

8.4.6.3 Download_Setup

8.4.6.3.1 Description

Ce service prépare le téléchargement d'un domaine, qui suit en plusieurs segments.

Si un intervalle de plus de 16 s s'écoule entre les chargements consécutifs des deux domaines, l'Agent doit réinitialiser la station.

8.4.6.3.2 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 la Station à démarrer
                                                 le programme de
                                                 téléchargement. Les autres
                                                 paramètres sont ignorés.
    DNLD_CHECK_ONLY,        (1)               -- vérifie la validité des
                                                 paramètres de téléchargement
                                                 (banc de mémoire, partition,
                                                 adresses et taille) mais
                                                 sans affecter la station.
    DNLD_START_ERASE        (2),              -- si les paramètres sont
                                                 valides, désactive les
                                                 domaines, efface la mémoire
                                                 et prépare le
                                                 téléchargement.
    DNLD_START_NOERASE      (3),              -- si les paramètres sont
                                                 valides, désactive les
                                                 domaines et prépare le
                                                 téléchargement.
    DNLD_TERMINATE_BOOT     (4),              -- termine le téléchargement et
                                                 redémarre
    DNLD_TERMINATE_NOBOOT (5),                -- termine le téléchargement,
                                                 arrête le temporisateur et
                                                 attend d'autres appels de
                                                 service.
    DNLD_VERIFY             (6)               -- appelle la procédure de
                                                 vérification pour ce
                                                 domaine.
    },
  reserved2              WORD8 (=0)
  download_time_out      UNSIGNED8,           -- temps admis entre deux
                                                 chargements de segment
                                                 (maximum 16 s).
  reserved3              WORD8 (=0)
  nr_domains             UNSIGNED8,           -- nombre de domaines à
                                                 préparer
  domain_list            ARRAY [nr_domains] OF
    {
    base_address         WORD32,              -- adresse de base du domaine
                                                 dans l'espace d'adresse de
                                                 l'Agent.
```

```
    domain_size         WORD32              -- taille du domaine en octets.
    }
  }
```

8.4.6.3.3 Reply_Write_Download_Setup

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															
premier setup_result								second setup_result							
dernier setup_result si nr_domains est impair								dernier setup_result si nr_domains est pair, sinon ‘00’H							

```
Reply_Write_Download_Setup::= RECORD
  {
  max_segment_size      UNSIGNED32,         -- taille maximale du tampon de
                                               téléchargement
  reserved1             WORD8 (=0),
  nr_domains            UNSIGNED8,          -- copie de ‘nr_domains’ dans
                                               l’appel
  setup_result_list     ARRAY [nr_domains] OF
    setup_result        ENUM8
    {
    DOMAIN_OK           (0),                -- domaine prêt au
                                               téléchargement
    DOMAIN_BAD_BASE_ADDR  (1),              -- adresse de base du domaine
                                               incorrecte
    DOMAIN_BAD_SIZE     (2),                -- taille de domaine incorrecte
    DOMAIN_ERASE_ERR    (3),                -- domaine ne peut être effacé
    DOMAIN_WRITE_ERR    (4),                -- domaine ne peut pas être
                                               écrit
    DOMAIN_BAD_CHECKSUM (5)                 -- somme de contrôle incorrecte
    }
  }
```

8.4.6.4 Download_Segment

8.4.6.4.1 Description

Ce service transmet un segment de taille définie à un domaine ouvert par Write_Download_Setup.

8.4.6.4.2 Call_Write_Download_Segment

0–7	8–15
tnm_key	sif_code = 55
reserved1	domain_id
segment_base_address	
segment_size	
segment_values: ARRAY [segment_size] OF	
premier octet à une adresse paire ou ‘00’H	second octet ou premier octet à une adresse impaire
octet_element	
dernier ou avant dernier octet	dernier octet ou ‘00’H

```
Call_Write_Download_Segment::= RECORD
  {
  reserved1             WORD8 (=0),          -- remplissage
  domain_id             UNSIGNED8,           -- identifie le domaine (index
                                                du domaine dans la
                                                ‘domain_list’ du dernier
                                                Call_Write_Download_Setup)
  segment_base_address  UNSIGNED32,          -- adresse de base du segment
                                                (peut être impaire)
  segment_size          UNSIGNED32,          -- taille du segment en octets
  segment_values        ARRAY [segment_size] OF
    {
    octet_element       WORD8                -- liste d’octets
    }
  }
```

8.4.6.4.3 Reply_Write_Download_Segment

0–7	8–15
tnm_key	sif_code = 55

```
Reply_Write_Download_Segment::= RECORD
  {}                                         -- aucun paramètre
```

8.4.7 Services de tâche

8.4.7.1 Read_Tasks_Status

8.4.7.1.1 Description

Ce service extrait le nom et l’état des tâches installées sur une station.

8.4.7.1.2 Call_Read_Tasks_Status

0–7	8–15
tnm_key	sif_code = 60

```
Call_Read_Tasks_Status::= RECORD
  {}                                       -- aucun paramètre
```

8.4.7.1.3 Reply_Read_Tasks_Status

0 … 7	8 … 15
tnm_key	sif_code = 60
reserved1	nr_tasks
tasks_list: ARRAY [nr_tasks] OF	
task_name: STRING16	
	CHARACTER8 or ‘00’H
priorité	état
cpu_load	
stack_margin	
task_comment: STRING26	
	CHARACTER8 or ‘00’H

```
Reply_Read_Tasks_Status::= RECORD
  {
  reserved1            WORD8 (=0),         -- remplissage
  nr_tasks             UNSIGNED8,          -- nombre de descripteurs
                                              d’état de tâche renvoyés
  tasks_list           ARRAY [nr_tasks] OF
    {
    task_name          STRING16,           -- nom ou numéro de tâche sous
                                              forme de chaîne
    priority           UNSIGNED8,          -- priorité de la tâche (0 =
                                              priorité maximale)
    status             ENUM8               -- état de la tâche
      {
      READY     (0),
      SUSPENDED (1),
      PENDING   (2),
      RUNNING   (3),
      FAULTY    (4)
      },
    cpu_load           UNSIGNED16,         -- charge du processeur générée
                                              par cette tâche en pourcent
                                              (0..100 %);
                                              les autres valeurs indiquent
                                              que la mesure de la charge
                                              du processeur n’est pas
                                              prise en charge
                                              stack_margin   UNSIGNED16,
                                               --   marge de pile de
                                              mémoire (ou ‘FFFF’H si ce
                                              service n’est pas fourni)
    task_comment       STRING26
    }
  }
```

8.4.7.2 Write_Tasks_Control

8.4.7.2.1 Description

Démarre ou arrête toutes les tâches.

Le bit "dnr" (Device Not Ready) doit être activé dans le Station_Status et le Device_Status de toutes les couches de liaison du MVB lorsqu'un arrêt est demandé, puis désactivé après le démarrage réussi.

8.4.7.2.2 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)                     -- arrête toutes les tâches
    START_TASK    (1)                     -- démarre toutes les tâches
    }
  task_id                UNSIGNED8        -- identifie la tâche (index de
                                             cette tâche dans la
                                             tasks_list de
                                             Reply_Read_Tasks_Status) à
                                             démarrer ou arrêter, 'FF'H
                                             démarre/arrête toutes les
                                             tâches
  }
```

8.4.7.2.3 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
  {}                                      -- aucun paramètre
```

8.4.8 Services d'horloge

8.4.8.1 Read_Clock

8.4.8.1.1 Description

Lit la valeur de l'horloge sur la station sélectionnée.

8.4.8.1.2 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
  {}                                      -- aucun paramètre
```

8.4.8.1.3 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 (secondes) time_date (ticks)															

```
Reply_Read_Clock::= RECORD
  {
  réservé              WORD16 (=0)          -- pour l'alignement
  time_date            TIMEDATE48           -- valeur du temps en secondes
                                               et ticks
  }
```

8.4.8.2 Write_Clock

8.4.8.2.1 Description

Met à l'heure l'horloge de la station sélectionnée.

8.4.8.2.2 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 (secondes) time_date (ticks)															

```
Call_Write_Clock::= RECORD
  {
  reserved1            WORD16 (=0)          -- pour alignement
pour l'alignement
  time_date            TIMEDATE48           -- valeur du temps en secondes
                                               et en ticks
  }
```

8.4.8.2.3 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
  {}                                        -- aucun paramètre
```

8.4.9 Service de journal

8.4.9.1 Read_Journal

8.4.9.1.1 Description

Ce service lit les dernières entrées "j" enregistrées dans le journal. La signification des entrées dépend de l'application. Le traitement de l'index est indiqué dans la description d'objet.

8.4.9.1.2 Call_Read_Journal

0 … 7	8 … 15
tnm_key	sif_code = 80
reserved1	number_entries

```
Call_Read_Journal::= RECORD
  {
  reserved1               WORD8 (=0),             -- réservé
  number_entries          UNSIGNED8               -- jusqu'à 255 entrées.
  }
```

8.4.9.1.3 Reply_Read_Journal

0 … 7	8 … 15
tnm_key	sif_code = 80
reserved1	number_entries
event_list ARRAY [number_entries] OF	
time_stamp: TIMEDATE48	
file_name: STRING16	
(CHARACTER8)	'00'H
line_number	
reserved2	event_type
event_description: STRING78	
(CHARACTER8)	'00'H

```
Reply_Read_Journal::= RECORD
  {
  reserved1               WORD8 (=0),
  number_entries          UNSIGNED8,              -- nombre d'entrées renvoyées
  event_list              ARRAY [number_entries] OF
    {
    time_stamp            TIMEDATE48,             -- estampille de temps d'un
                                                     événement
    file_name             STRING16,               -- comme obtenu par __FILE__ en
                                                     ANSI 'C' (chaîne terminée
                                                     par un zéro)
    line_number           UNSIGNED16,             -- comme fourni par __LINE__ en
                                                     ANSI 'C'
    reserved2             WORD8 (=0),
    event_type            ENUM8                   -- type d'événement
      {
      INFO              (0),
      WARNING           (1),
      ERROR             (2)
      },
    event_description   STRING78                  -- description de l'événement
                                                     (chaîne terminée par zéro)
    }
  }
```

8.4.10 Service d'Equipement

8.4.10.1 Read_Equipment

8.4.10.1.1 Description

Ce service récupère un pointeur vers le domaine de mémoire dans lequel se trouve une description complète de l'équipement pris en charge. Le format de cette structure de données n'entre pas dans le domaine d'application de la présente Norme.

8.4.10.1.2 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),          -- réservé
  reserved2             WORD8 (=0),          -- réservé
  }
```

8.4.10.1.3 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,           -- nombre d'entrées renvoyées
  equipment_list        ARRAY [number_entries] OF
    {
    equipment_name      STRING32,            -- identifie l'équipement
    equipment_root      UNSIGNED32           -- adresse de base du domaine
    equipment_size      UNSIGNED32           -- taille du descripteur
                                                d'équipement
    }
  }
```

8.5 Procédures d'interface

Les procédures d'interface sont divisées entre une interface du Gestionnaire et une interface de l'agent.

8.5.1 Interface du Gestionnaire (MGI)

Tous les services de l'interface du Gestionnaire sont fournis par deux procédures génériques:

- une procédure de demande de service mm_service_req, et
- une procédure de confirmation de service mm_service_conf.

Les procédures d'interface du Gestionnaire ont le préfixe mm_xxx.

Description	Appelle un service à distance.	

Syntaxe	`MM_RESULT`
	`UNSIGNED8`
	`const AM_ADDRESS*`
	`struct MM_CALL *`
		`mm_service_req`
		`(`
		`station_id;`
		`agent_adr;`
		`mm_call`
		`);`

Entrée	station_id	Station_Identifier de cette Station
	agent_adr	Network_Address de l’Agent
	mm_call	mm_call identique au format du corps du Call_Message de Gestion (le format de cette structure dépend du SIF_code)
Résultat		le résultat de l’appel est un code d’erreur MM_RESULT (voir 8.4.1.5).

Description	La procédure de confirmation de service mm_service_conf renvoie le résultat de l’appel de service au Gestionnaire. Cette procédure peut être interrogée ou peut être une procédure d’indication.	

Syntaxe	`MM_RESULT`
	`UNSIGNED8`
	`AM_ADDRESS *`
	`struct MM_REPLY *`
		`mm_service_conf`
		`(`
		`station_id;`
		`agent_adr;`
		`mm_reply`
		`);`

Sortie	station_id	Station_Identifier de cette Station
	agent_adr	Network_Address de l’Agent
	mm_reply	si MM_RESULT est OK, cette structure renvoie le corps de Reply_Message, sinon, elle n’est pas définie (le format de cette structure dépend du SIF_code)
Résultat		le résultat de l’appel est un code d’erreur MM_RESULT (voir XXX).

8.5.2 Interface de l’Agent

8.5.2.1 Description

Les procédures d’interface de l’Agent ne définissent pas l’interface entre l’Agent et le réseau, mais entre l’Agent et les autres processus de la Station.

L’interface de l’Agent permet à l’utilisateur d’accéder à l’Agent, pour interroger deux conditions:

- change (est-il possible d’apporter des modifications à la Station ?)
- stop (est-il possible d’arrêter la station ?).

L'Agent peut avoir besoin d'accéder au noyau en temps réel pour coordonner l'exécution des tâches de l'utilisateur.

Les procédures d'interface de l'Agent ont le préfixe ma_xxx.

8.5.2.2 Procédures de contrôle de l'Agent

8.5.2.2.1 Procédure ma_ask_permission

Description	Permet à un utilisateur de savoir quelles demandes de gestion existent. L'application qui utilise cette fonction répond par ma_permit.		

Syntaxe	`MA_PERMISSION`
	
	`UNSIGNED8`
		`ma_ask_permission`
		`(`
		`task_id`
		`);`	

Entrée	task_id	appel de la tâche (identifié par un index de la tâche dans la tasks_list Reply_Read_Tasks_Status)	
Sortie	-		
Renvoi	MA_PERMISSION	0: MA_CHANGE_REQU, 1: MA_CHANGE_NOREQU, 2: MA_STOP_REQU, 3: MA_STOP_NOREQU	modifications requises aucune modification requise arrêt requis aucun arrêt requis

8.5.2.2.2 Procédure ma_give_permission

Description	L'application répond à la demande de changement avec cette procédure, en indiquant si le changement est permis ou non.		

Syntaxe	`void`
	
	`ENUM8`
		`ma_give_permission`
		`(`
		`decision`
		`);`	

Entrée	decision	0: MA_CHANGE_ALLOWED, 1: MA_CHANGE_DENIED, 2: MA_STOP_ALLOWED, 3: MA_STOP_DENIED	changement permis changement interdit arrêt permis arrêt interdit
Sortie	-		

8.5.2.3 Souscription d'un service utilisateur

8.5.2.3.1 Type MA_SERVICE_CALL

Description	Déclaration du type d'une procédure à appeler pour un certain appel de service, qui renvoie les paramètres nécessaires pour le Reply_Message.	

Syntaxe	`typedef void`
	`AM_ADDRESS *`
	`void *`
	`UNSIGNED32`
	`void * *`
	`UNSIGNED32 *`
	`MM_RESULT *`
		`( * MA_SERVICE_CALL )`
		`(`
		`manager_address,`
		`call_msg_adr,`
		`call_msg_size,`
		`reply_msg_adr,`
		`reply_msg_size`
		`agent_status`
		`);`

Entrée	manager_address	pointeur vers l'adresse de réseau complète du gestionnaire d'appel
	call_msg_adr	pointeur vers le début du Call_Message de service à traiter (champ tnm_key)
	call_msg_size	taille du Call_Message en octets
	reply_msg_adr	pointeur vers le début du Reply_Message du service (champ tnm_key) à renvoyer
	reply_msg_size	taille du Call_Message en octets
	agent_status	résultat à communiquer comme état de l'Agent au Gestionnaire

8.5.2.3.2 Type MA_SERVICE_CLOSE

Description	Déclaration du type de la procédure à appeler pour clore un service.	
Syntaxe	`typedef void`	`( * MA_SERVICE_CLOSE )` `(void);`
Entrée	undefined	défini par l'utilisateur

8.5.2.3.3 Procédure ma_subscribe

Description	Indique quelle procédure utilisateur à appeler en cas de réception d'un appel de service défini par l'utilisateur. Un SIF_code préalablement assigné est écrasé sans avertissement	

Syntaxe	`MM_RESULT`
	`ENUM16`
	`ENUM8`
	`MA_SERVICE_CALL`
	`MA_SERVICE_CLOSE`
	`void *`
		`ma_subscribe`
		`(`
		`command;`
		`sif_code;`
		`service_call;`
		`service_close`
		`service_desc`
		`);`

Entrée	command	0: souscrit 1: résilie la souscription
	sif_code	SIF_code (≥ 128) de l'utilisateur
	service_call	variable de procédure du type MA_SERVICE_CALL qui exécute le service lorsqu'il est appelé
	service_close	procédure que l'Agent appelle lorsque la totalité du Reply_Message a été envoyée (pour libérer les tampons, par exemple).
	service_desc	Descripteur du service, sous la forme d'une chaîne visible terminée par un caractère '00'H.
Renvoi	MM_RESULT	

8.5.2.4 Souscription de la procédure de redémarrage

8.5.2.4.1 Type MA_STATION_RESTART

Description	Déclaration du type de la procédure à appeler pour redémarrer la Station après une temporisation ou une commande de redémarrage. Cette procédure ne sera probablement pas retournée.	
Syntaxe	`typedef void`	`( * MA_STATION_RESTART )` `( );`

8.5.2.4.2 Procédure ma_subscribe_restart

Description	Indique la procédure de l'utilisateur à appeler au redémarrage de la Station ou à l'échéance de la temporisation de réservation	

Syntaxe	`MM_RESULT`
	`MA_STATION_RESTART`
		`ma_subscribe_restart`
		`(`
		`station_restart`
		`);`

Entrée	station_restart	procédure que l'Agent appelle à l'échéance de la temporisation de réservation ou à la réception d'une commande de réinitialisation
Renvoi	MM_RESULT	

Bibliographie

CEI 60870-5-1, *Matériels et systèmes de téléconduite – Partie 5: Protocoles de transmission – Première Section: Formats de trames de transmission*

ISO/CEI 8482, *Technologies de l'information – Télécommunications et échange d'information entre systèmes – Interconnexions multipoints par paire torsadée*

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

Tel: + 41 22 919 02 11
Fax: + 41 22 919 03 00
info@iec.ch
www.iec.ch