IEC 61158-6-2

Edition 3.0 2014-08

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Industrial communication networks – Fieldbus specifications – Part 6-2: Application layer protocol specification – Type 2 elements

Réseaux de communication industriels – Spécifications des bus de terrain – Partie 6-2: Spécification du protocole de la couche application – Eléments de type 2

IEC 61158-6-2:2014-08(en-fr)

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.

IEC Catalogue - webstore.iec.ch/catalogue
The stand-alone application for consulting the entire bibliographical information on IEC International Standards, Technical Specifications, Technical Reports and other documents. Available for PC, Mac OS, Android Tablets and iPad.

IEC publications search - www.iec.ch/searchpub
The advanced search enables 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 It also gives information on projects, replaced and withdrawn publications.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 online 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 14 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary
More than 55 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

IEC 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 l'IEC
La Commission Electrotechnique Internationale (IEC) 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 IEC
Le contenu technique des publications IEC est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Catalogue IEC - webstore.iec.ch/catalogue
Application autonome pour consulter tous les renseignements bibliographiques sur les Normes internationales, Spécifications techniques, Rapports techniques et autres documents de l'IEC. Disponible pour PC, Mac OS, tablettes Android et iPad.

Recherche de publications IEC - www.iec.ch/searchpub
La recherche avancée permet de trouver des publications IEC 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.

IEC Just Published - webstore.iec.ch/justpublished
Restez informé sur les nouvelles publications IEC. 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 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 14 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

Glossaire IEC - std.iec.ch/glossary
Plus de 55 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

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 61158-6-2

Edition 3.0 2014-08

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Industrial communication networks – Fieldbus specifications – Part 6-2: Application layer protocol specification – Type 2 elements

Réseaux de communication industriels – Spécifications des bus de terrain – Partie 6-2: Spécification du protocole de la couche application – Eléments de type 2

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE XH
CODE PRIX

ICS 25.040.40; 35.100.70; 35.110

ISBN 978-2-8322-1756-6

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-2: Application layer protocol specification – Type 2 elements

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC 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.

Attention is drawn to the fact that the use of the associated protocol type is restricted by its intellectual-property-right holders. In all cases, the commitment to limited release of intellectual-property-rights made by the holders of those rights permits a layer protocol type to be used with other layer protocols of the same type, or in other type combinations explicitly authorized by its intellectual-property-right holders.

NOTE Combinations of protocol types are specified in IEC 61784-1 and IEC 61784-2.

International Standard IEC 61158-6-2 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

This third edition cancels and replaces the second edition published in 2010. This edition constitutes a technical revision.

The main changes with respect to the previous edition are listed below:

- Updates of definition used by the Time Sync object;
- Addition of “member” and object specific services in 4.1.2.1, 4.1.8, 4.1.10, 8.2;
- Removal of obsolete transport classes 4 to 6 in 4.1.4.6, and 9.3.9 to 9.3.12;
- Clarification of transport header formats in 4.1.4;
- Update of CM and MR PDUs in 4.1.5 to 4.1.7;
- Updates of Identity object PDUs in 4.1.8.2;
- Updates of Assembly object PDUs in 4.1.8.4;
- Updates of Time sync object PDUs in 4.1.8.6;
- Updates of Parameter object PDUs in 4.1.8.7;
- Updates of Connection Manager object PDUs in 4.1.8.8;
- Updates of message and connection paths in 4.1.9;
- Updates of object class codes in 4.1.10 and error codes in 4.1.11;
- Updates of data types in 4.2.4 and 5.2.3;
- Updates of the encapsulation abstract syntax in 4.3;
- Updates to the DLL mapping protocol machine 2 in Clause 11;
- Miscellaneous editorial corrections.

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

FDIS	Report on voting
65C/764/FDIS	65C/774/RVD

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

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

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

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.

INTRODUCTION

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

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

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

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

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-2: Application layer protocol specification – Type 2 elements

1 Scope

1.1 General

The Fieldbus Application Layer (FAL) provides user programs with a means to access the fieldbus communication environment. In this respect, the FAL can be viewed as a “window between corresponding application programs.”

This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 2 fieldbus. The term “time-critical” is used to represent the presence of a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

This standard specifies interactions between remote applications and defines the externally visible behavior provided by the Type 2 fieldbus application layer in terms of

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

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

a) define the wire-representation of the service primitives defined in IEC 61158-5-2, and
b) define the externally visible behavior associated with their transfer.

This standard specifies the protocol of the Type 2 fieldbus application layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498-1) and the OSI application layer structure (ISO/IEC 9545).

1.2 Specifications

The principal objective of this standard is to specify the syntax and behavior of the application layer protocol that conveys the application layer services defined in IEC 61158-5-2.

A secondary objective is to provide migration paths from previously-existing industrial communications protocols.

1.3 Conformance

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

2 Normative references

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

NOTE All parts of the IEC 61158 series, as well as IEC 61784-1 and IEC 61784-2 are maintained simultaneously. Cross-references to these documents within the text therefore refer to the editions as dated in this list of normative references.

IEC 61158-1:2014, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61158-3-2:2014, *Industrial communication networks – Fieldbus specifications – Part 3-2: Data-link layer service definition – Type 2 elements*

IEC 61158-4-2:2014, *Industrial communication networks – Fieldbus specifications – Part 4-2: Data-link layer protocol specification – Type 2 elements*

IEC 61158-5-2:2014, *Industrial communication networks – Fieldbus specifications – Part 5-2: Application layer service definition – Type 2 elements*

IEC 61588:2009, *Precision clock synchronization protocol for networked measurement and control systems*

IEC 61784-3-2, *Industrial communication networks – Profiles – Part 3-2: Functional safety fieldbuses – Additional specifications for CPF 2*

IEC 61800-7-202, *Adjustable speed electrical power drive systems – Part 7-202: Generic interface and use of profiles for power drive systems – Profile type 2 specification*

IEC 62026-3:2008, *Low-voltage switchgear and controlgear – Controller-device interfaces (CDIs) – Part 3: DeviceNet*

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

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

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

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

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

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

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

ISO 639-2, *Codes for the representation of names of languages – Part 2: Alpha-3 code*

ISO 11898:1993[1], *Road vehicles – Interchange of digital information – Controller area network (CAN) for high-speed communication*

IEEE 802.1D-2004, *IEEE standard for local and metropolitan area networks – Media Access Control (MAC) bridges*, available at http://www.ieee.org

IEEE 802.1Q-2005[1], *IEEE standard for local and metropolitan area networks – Virtual bridged local area networks*, available at <http://www.ieee.org>

IEEE 802.3-2008: *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications,* available at <http://www.ieee.org>

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

IETF RFC 1035, *Domain Names – Implementation and Specification*, available at <http://www.ietf.org>

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

IETF RFC 1117, *Internet Numbers*, available at <http://www.ietf.org>

IETF RFC 1122, *Requirements for Internet Hosts – Communication Layers*, available at <http://www.ietf.org>

IETF RFC 1759, *Printer MIB*, available at <http://www.ietf.org>

IETF RFC 2236, *Internet Group Management Protocol, Version 2*, available at <http://www.ietf.org>

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

IETF RFC 2475, *An Architecture for Differentiated Services*, available at <http://www.ietf.org>

IETF RFC 2597, *Assured Forwarding PHB Group*, available at <http://www.ietf.org>

IETF RFC 2873, *TCP Processing of the IPv4 Precedence Field*, available at <http://www.ietf.org>

IETF RFC 3140, *Per Hop Behavior Identification Codes*, available at <http://www.ietf.org>

---

[1] A newer edition of this standard has been published, but only the cited edition applies.

IETF RFC 3246, *An Expedited Forwarding PHB (Per-Hop Behavior)*, available at <http://www.ietf.org>

IETF RFC 3376, *Internet Group Management Protocol, Version 3*, available at <http://www.ietf.org>

IETF RFC 4594, *Configuration Guidelines for DiffServ Service Classes*, available at <http://www.ietf.org>

3 Terms, definitions, symbols, abbreviations and conventions

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

3.1 Terms and definitions from other ISO/IEC standards

3.1.1 Terms and definitions from ISO/IEC 7498-1

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

3.1.2 Terms and definitions from ISO/IEC 9545

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

3.1.3 Terms and definitions from ISO/IEC 8824-1

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

t) module
u) production

3.1.4 Terms and definitions from ISO/IEC 8825-1

a) encoding (of a data value)
b) data value
c) identifier octets (the singular form is used in this standard)
d) length octet(s) (both singular and plural forms are used in this standard)
e) contents octets

3.2 Terms and definitions from IEC 61158-5-2

a) application relationship
b) client
c) peer
d) server

3.3 Additional terms and definitions

3.3.1
allocate
take a resource from a common area and assign that resource for the exclusive use of a specific entity

3.3.2
application
function or data structure for which data is consumed or produced

3.3.3
application objects
multiple object classes that manage and provide a run time exchange of messages across the network and within the network device

3.3.4
attribute
description of an externally visible characteristic or feature of an object

Note 1 to entry: The attributes of an object contain information about variable portions of an object. Typically, they provide status information or govern the operation of an object. Attributes may also affect the behavior of an object. Attributes are divided into class attributes and instance attributes.

3.3.5
behavior
indication of how an object responds to particular events

3.3.6
Best Master Clock Algorithm
BMCA
algorithm performed by each node to determine the clock that will become the master clock on a subnet and the grandmaster clock for the domain

Note 1 to entry: The algorithm primarily compares priority1, clock quality, priority2, and source identity to determine the best master among available candidates.

3.3.7
boundary clock
clock that has multiple Precision Time Protocol (PTP) ports in a domain and maintains the timescale used in the domain

Note 1 to entry: It may serve as the source of time, i.e., be a master clock, and may synchronize to another clock, i.e., be a slave clock.

[SOURCE: IEC 61588:2009, 3.1.3, modified – second sentence changed to a Note]

3.3.8
called
service user or a service provider that receives an indication primitive or a request APDU

3.3.9
calling
service user or a service provider that initiates a request primitive or a request APDU

3.3.10
class
set of objects, all of which represent the same kind of system component

Note 1 to entry: A class is a generalization of an object; a template for defining variables and methods. All objects in a class are identical in form and behavior, but usually contain different data in their attributes.

3.3.11
class attribute
attribute that is shared by all objects within the same class

3.3.12
class code
unique identifier assigned to each object class

3.3.13
class specific service
service defined by a particular object class to perform a required function which is not performed by a common service

Note 1 to entry: A class specific object is unique to the object class which defines it.

3.3.14
client
a) object which uses the services of another (server) object to perform a task

b) initiator of a message to which a server reacts

3.3.15
clock
node participating in the Precision Time Protocol (PTP) that is capable of providing a measurement of the passage of time since a defined epoch

Note 1 to entry: There are three types of clocks in IEC 61588:2009, boundary, transparent and ordinary clocks.

[SOURCE: IEC 61588:2009, 3.1.4, modified – different Note]

3.3.16
communication objects
components that manage and provide a run time exchange of messages across the network

EXAMPLES Connection Manager object, Unconnected Message Manager (UCMM) object, and Message Router object.

3.3.17
connection
logical binding between application objects that may be within the same or different devices

Note 1 to entry: Connections may be either point-to-point or multipoint.

3.3.18
connection ID
CID
identifier assigned to a transmission that is associated with a particular connection between producers and consumers, providing a name for a specific piece of application information

3.3.19
connection path
octet stream that defines the application object to which a connection instance applies

3.3.20
connection point
buffer which is represented as a subinstance of an Assembly object

3.3.21
consume
act of receiving data from a producer

3.3.22
consumer
node or sink that is receiving data from a producer

3.3.23
consuming application
application that consumes data

3.3.24
cyclic
repetitive in a regular manner

3.3.25
device
physical hardware connected to the link

Note 1 to entry: A device may contain more than one node.

3.3.26
device profile
collection of device dependent information and functionality providing consistency between similar devices of the same device type

3.3.27
domain
logical grouping of clocks that synchronize to each other using the protocol, but that are not necessarily synchronized to clocks in another domain

[SOURCE: IEC 61588:2009, 3.1.7]

3.3.28
end node
producing or consuming node

3.3.29
end point
one of the communicating entities involved in a connection

3.3.30
epoch
origin of a time scale

[SOURCE: IEC 61588:2009, 3.1.9]

3.3.31
error
discrepancy between a computed, observed or measured value or condition and the specified or theoretically correct value or condition

3.3.32
frame
denigrated synonym for DLPDU

3.3.33
grandmaster clock
within a domain, clock that is the ultimate source of time for clock synchronization using the PTP protocol

[SOURCE: IEC 61588:2009, 3.1.13]

3.3.34
instance
actual physical occurrence of an object within a class that identifies one of many objects within the same object class.

EXAMPLE California is an instance of the object class state.

Note 1 to entry: The terms object, instance, and object instance are used to refer to a specific instance.

3.3.35
instance attribute
attribute that is unique to an object instance and not shared by the object class

3.3.36
instantiated
object that has been created in a device

3.3.37
interoperability
capability of User Layer entities to perform coordinated and cooperative operations using the services of the FAL

3.3.38
Keeper
object responsible for distributing link configuration data to all nodes on the link

3.3.39
little endian
model of memory organization which stores the least significant octet at the lowest address, or for transfer, which transfers the lowest order octet first

Note 1 to entry: Native Type 2 data types are sent in little endian order.

3.3.40
Lpacket
Link packet
piece of application information that contains a size, control octet, tag, and link data

Note 1 to entry: Peer data-link layers use Lpackets to send and receive service data units from higher layers in the OSI stack.

3.3.41
management information
network accessible information that supports managing the operation of the fieldbus system, including the application layer

Note 1 to entry: Managing includes functions such as controlling , monitoring, and diagnosing.

3.3.42
master clock
in the context of a single Precision Time Protocol (PTP) communication path, clock that is the source of time to which all other clocks on that path synchronize

[SOURCE: IEC 61588:2009, 3.1.17]

3.3.43
member
piece of an attribute that is structured as an element of an array

3.3.44
Message Router
object within a node that distributes messaging requests to appropriate application objects

3.3.45
multipoint connection
connection from one node to many

Note 1 to entry: Multipoint connections allow messages from a single producer to be received by many consumer nodes.

3.3.46
network
set of nodes connected by some type of communication medium, including any intervening repeaters, bridges, routers and lower-layer gateways

3.3.47
object
abstract representation of a particular component within a device, usually a collection of related data (in the form of variables) and methods (procedures) for operating on that data that have clearly defined interface and behavior

3.3.48
object specific service
service unique to the object class which defines it

3.3.49
ordinary clock
clock that has a single Precision Time Protocol (PTP) port in a domain and maintains the timescale used in the domain

Note 1 to entry: It may serve as a source of time, i.e., be a master clock, or may synchronize to another clock, i.e., be a slave clock.

[SOURCE: IEC 61588:2009, 3.1.22, modified – second sentence changed to a Note]

3.3.50
originator
client responsible for establishing a connection path to the target

3.3.51
parent clock
master clock to which a clock is synchronized

[SOURCE: IEC 61588:2009, 3.1.23]

3.3.52
point-to-point connection
connection that exists between exactly two application objects

3.3.53
Precision Time Protocol
PTP
protocol defined by IEC 61588:2009

Note 1 to entry: As an adjective, it indicates that the modified noun is specified in or interpreted in the context of IEC 61588:2009.

[SOURCE: IEC 61588:2009, 3.1.28, modified – second sentence changed to a Note]

3.3.54
produce
act of sending data to be received by a consumer

3.3.55
producer
node that is responsible for sending data

3.3.56
PTP message
one of the message types defined in IEC 61588:2009

[SOURCE: IEC 61588:2009, 3.1.33]

3.3.57
PTP port
logical access point of a clock for PTP communications to the communications network

[SOURCE: IEC 61588:2009, 3.1.35]

3.3.58
receiving
service user that receives a confirmed primitive or an unconfirmed primitive, or a service provider that receives a confirmed APDU or an unconfirmed APDU

3.3.59
resource
processing or information capability of a subsystem

3.3.60
sending
service user that sends a confirmed primitive or an unconfirmed primitive, or a service provider that sends a confirmed APDU or an unconfirmed APDU

3.3.61
server
a) role of an AREP in which it returns a confirmed service response APDU to the client that initiated the request

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

3.3.62
service
operation or function than an object and/or object class performs upon request from another object and/or object class

3.3.63
synchronized clocks
(to a specified uncertainty) two clocks which have the same epoch and for which measurements of the time of a single event at an arbitrary time differ by no more than the specified uncertainty

[SOURCE: IEC 61588:2009, 3.1.40, modified – reworded]

3.3.64
System Time
absolute time value as defined by CPF2 time synchronization in the context of a distributed time system where all devices have a local clock that is synchronized with a common master clock

Note 1 to entry: In the context of CPF2, System Time is a 64-bit integer value in units of nanoseconds with a value of 0 corresponding to the date 1970-01-01.

3.3.65
target
end-node to which a connection is established

3.3.66
temporary node
transient node

3.3.67
transaction id
field within a UCMM header that matches a response with the associated request

3.3.68
transparent clock
device that measures the time taken for a Precision Time Protocol (PTP) event message to transit the device and provides this information to clocks receiving this PTP event message

[SOURCE: IEC 61588:2009, 3.1.46, modified – truncated]

3.3.69
Unconnected Message Manager
UCMM
component within a node that transmits and receives unconnected explicit messages and sends them directly to the Message Router object

3.3.70
unconnected service
messaging service which does not rely on the set up of a connection between devices before allowing information exchanges

3.3.71
vendor ID
identification of each product manufacturer/vendor by a unique number

Note 1 to entry: Vendor IDs are assigned by ODVA, Inc.

3.4 Abbreviations and symbols

ASCII	American Standard Code for Information Interchange
CID	connection ID
DLL	data-link layer
DSCP	DiffServ Codepoint
IGMP	Internet Group Management Protocol (see IETF RFC 1112, IETF RFC 2236)
IP	Internet Protocol (see IETF RFC 791)
IPv4	Internet Protocol version 4 (see IETF RFC 791)
IPv6	Internet Protocol version 6 (see IETF RFC 791)
OSI	open systems interconnection (see ISO/IEC 7498-1)
PDU	protocol data unit
PHB	per-hop behavior
PTP	Precision Time Protocol (see IEC 61588:2009)
QoS	quality of service
Rcv	receive
Rx	receive
SDU	service data unit
SEM	state event matrix
STD	state transition diagram, used to describe object behavior
TCP	Terminal Control Protocol (see IETF RFC 793)
ToS	type of service
TPDU	transport protocol data unit
Tx	transmit
UDP	User Datagram Protocol (see IETF RFC 768)
Xmit	transmit
CM_API	actual packet interval
O2T or O⇒T	originator to target (connection parameters)
CM_RPI	requested packet interval
TUI	table unique identifier
T2O or T⇒O	target to originator (connection parameters)

3.5 Conventions

3.5.1 General concept

The FAL is defined as a set of object-oriented ASEs. Each ASE is specified in a separate subclause. Each ASE specification is composed of three parts: its class definitions, its services, and its protocol specification. The first two are contained in IEC 61158-5-2. The protocol specification for each of the ASEs is defined in this standard.

The class definitions define the attributes of the classes supported by each ASE. The attributes are accessible from instances of the class using the Management ASE services specified in IEC 61158-5-2. The service specification defines the services that are provided by the ASE.

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

Bold font is used in this standard to highlight parameter names or important requirement elements from surrounding text. Italic font is also used in Tables and Notes for that purpose for better visibility.

3.5.2 Attribute specification

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

Attribute ID	Name	Data type	Semantics of values

Figure 1 – Attribute table format and terms

The **Attribute ID** shall be a unique integer identification value assigned to an attribute. The valid ranges for Attribute IDs shall be as specified in 4.1.10.1.3. The Attribute ID shall identify the particular attribute being accessed.

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

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

NOTE The elementary data types are defined in IEC 61158-5-2.

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

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

3.5.3 Common services

3.5.3.1 Service_PDU definitions

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

Name	Type	Semantics of values

Figure 2 – Service request/response parameter

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

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

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

3.5.3.2 Get_Attribute_All response

3.5.3.2.1 General definition

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

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

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

Table 1 – Get_Attribute_All response service rules

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

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

Table 2 – Example class level object/service specific response data of Get_Attribute_All

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

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

Table 3 – Example Get_Attribute_All data array method

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

3.5.3.2.2 Revisions

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

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

3.5.3.3 Set_Attribute_All request

3.5.3.3.1 General definition

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

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

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

Table 4 – Set_Attribute_All request service rules

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

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

Table 5 – Example Set_Attribute_All attribute ordering method

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

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

Table 6 – Example Set_Attribute_All data array method

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

3.5.3.3.2 Revisions

A server processing a Set_Attribute_All request may need to process more data than is expected by the server if the client has implemented a newer revision of this object while the server an older revision. As a result, the server object may have to process a

Set_Attribute_All request that contains more data because the additional attributes were not recognized. If the server receives more data in a Set_Attribute_All request than it expects the server shall respond with a general status code equal to 0x15 (too much data).

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

3.5.4 State machine conventions

3.5.4.1 General

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

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

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

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

3.5.4.2 State Transition Diagram (STD)

An **event** is an external stimulus that may cause a state transition. An STD graphically illustrates the states of an object and includes events, service calls and changes of attributes that cause it to transition to another state. Figure 3 shows an example of an STD.

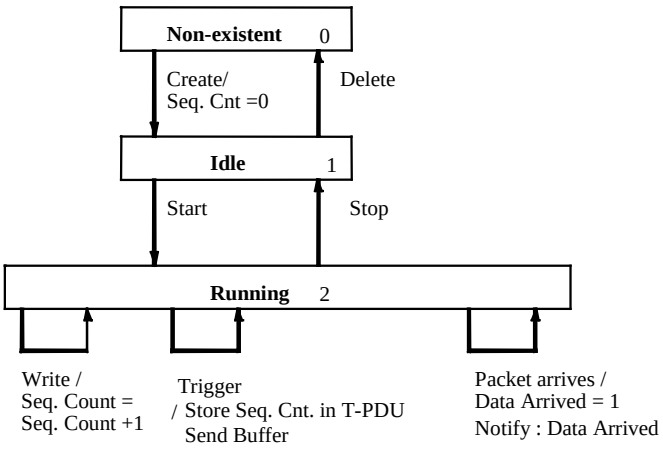


Figure 3 – Example of an STD

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

3.5.4.3 State Event Matrix

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

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

State event matrix format is as shown in Table 7.

Table 7 – State event matrix format

Event	State		
	State 1	....	State n
Event A description	Function triggered by event A in State 1 (if any) Notification to FAL user (if any) Transition to another state (if any)		Function triggered by event A in State n (if any) Notification to FAL user (if any) Transition to another state (if any)
.....	.....		.....
Event X description	Function triggered by event X in State 1 (if any) Notification to FAL user (if any) Transition to another state (if any)		Function triggered by event X in State n (if any) Notification to FAL user (if any) Transition to another state (if any)
In absence of function, notification or transition, the corresponding entry shall not be made in the table.			

3.5.4.4 Example state event matrix

Table 8 shows an example of a state machine with three states:

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

Table 8 – Example state event matrix

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

4 Abstract syntax

4.1 FAL PDU abstract syntax

4.1.1 General

FAL_PDU shall be either a UCMM_PDU or a Transport_PDU.

UCMM_PDU is used to convey information for connection-less services.

Transport_PDU is used to convey information for connection-oriented services.

A connected message assumes previously negotiated resources and parameters at its source, its destination(s), and any intermediate transit points. These resources are referenced by a unique connection identifier and do not need to be contained in each message, only the connection ID is needed to identify the message and refer to its related parameters, thus giving significant savings in message efficiency.

An unconnected message provides a means to communicate on the local link without previously negotiated resources at the destination so it shall carry full destination ID details, internal data descriptors and full source ID details if a reply is requested. Unconnected messages are used mainly to create connections.

The unconnected service is provided by the Unconnected Message Manager (UCMM). Messages received through the UCMM are forwarded to the Message Router (MR), which direct them to the appropriate internal object for execution. Connections are established by specific unconnected messages sent through the Connection Manager (CM) using UCMM services. Connections may be established either to the Message Router (for messaging purpose), or directly to an application object. Connection target is specified using a connection path, and may be either on the local or a remote link, through several intermediate router nodes. Once a connection is established with an application object, the UCMM, MR and CM are no longer required, since data will be exchanged directly with the connected objet, based on the corresponding connection ID. Connected messages sent to the Message Router will be forwarded to the appropriate internal object for execution.

4.1.2 PDU structure

4.1.2.1 UCMM_PDU structure

UCMM_PDU is a sequence of a UCMM_Header, followed by either OM_Service (Object Management ASE) or CM_Service (Connection Manager ASE).

OM_Service shall be either OM_Request or OM_Response.

OM_Request shall consist of MR_Request_Header and a Service_RequestPDU.

Service_RequestPDU shall be one of the following:

- Get_Attribute_Single_RequestPDU
- Get_Attribute_All_RequestPDU
- Get_Attribute_List_RequestPDU
- Set_Attribute_Single_RequestPDU
- Set_Attribute_All_RequestPDU
- Set_Attribute_List_RequestPDU
- Reset_RequestPDU
- Create_RequestPDU
- Delete_RequestPDU
- Start_RequestPDU
- Stop_RequestPDU
- Find_Next_Object_Instance_RequestPDU
- NOP_RequestPDU
- Apply_Attributes_RequestPDU

- Save_RequestPDU
- Restore_RequestPDU
- Get_Member_RequestPDU
- Set_Member_RequestPDU
- Insert_Member_RequestPDU
- Remove_Member_RequestPDU
- Group_Sync_RequestPDU
- Add_AckData_Path_RequestPDU
- Remove_AckData_Path_RequestPDU
- Get_Enum_String_RequestPDU
- Symbolic_Translation_RequestPDU
- Connection_Bind_RequestPDU
- Producing_Application_Lookup_RequestPDU

OM_Response shall consist of MR_Response_Header and a Service_ResponsePDU.

Service_ResponsePDU shall be one of the following:

- Get_Attribute_Single_ResponsePDU
- Get_Attribute_All_ResponsePDU
- Get_Attribute_List_ResponsePDU
- Set_Attribute_Single_ResponsePDU
- Set_Attribute_All_ResponsePDU
- Set_Attribute_List_ResponsePDU
- Reset_ResponsePDU
- Create_ResponsePDU
- Delete_ResponsePDU
- Start_ResponsePDU
- Stop_ResponsePDU
- Find_Next_Object_Instance_ResponsePDU
- NOP_ResponsePDU
- Apply_Attributes_ResponsePDU
- Save_ResponsePDU
- Restore_ResponsePDU
- Get_Member_ResponsePDU
- Set_Member_ResponsePDU
- Insert_Member_ResponsePDU
- Remove_Member_ResponsePDU
- Group_Sync_ResponsePDU
- Add_AckData_Path_ResponsePDU
- Remove_AckData_Path_ResponsePDU
- Get_Enum_String_ResponsePDU
- Symbolic_Translation_ResponsePDU
- Connection_Bind_ResponsePDU
- Producing_Application_Lookup_ResponsePDU

CM_Service shall be either CM_Request or CM_Response.

CM_Request shall consist of MR_Request_Header and a CM_RequestPDU.

CM_RequestPDU shall be one of the following:

- Forward_Open_RequestPDU
- Forward_Close_RequestPDU
- Large_Forward_Open_RequestPDU
- Unconnected_Send_RequestPDU
- Get_Connection_Data_RequestPDU
- Search_Connection_Data_RequestPDU
- Get_Connection_Owner_RequestPDU

CM_Response consists of MR_Response_Header and a CM_ResponsePDU.

CM_ ResponsePDU shall be one of the following:

- Forward_Open_ResponsePDU
- Forward_Close_ResponsePDU
- Large_Forward_Open_ResponsePDU
- Unconnected_Send_ResponsePDU
- Get_Connection_Data_ResponsePDU
- Search_Connection_Data_ResponsePDU
- Get_Connection_Owner_ResponsePDU

4.1.2.2 Transport_PDU structure

Transport_PDU is a sequence of a Transport_Header, followed by either OM_Service (Object Management ASE) or Application Data.

OM_Service is defined in 4.1.2.1 above.

Application Data comes from the source to which the connection has been made.

4.1.3 UCMM_PDUs

Subclause 4.1.3 defines the requirements for UCMM PDU's when using a Type 2 data-link layer.

All UCMM_PDUs shall be sent and received using fixed tag 0x83 or Management tag 0x88. When a device becomes powered, the UCMM shall invoke the `DLL_enable_fixed_request` service of the data-link layer to request that fixed tag 0x83 or 0x88 Lpackets be routed to the UCMM. All sending and receiving of TPDUs shall use the `DLL_fixed_request.req` and `DLL_fixed_request.ind` service primitives of the data-link layer, respectively. The packet parameter of these services shall be prefixed with the following header to form the Transport PDU before sending to the data-link layer. The format of the UCMM_PDU Header is shown in Table 9.

Table 9 – UCMM_PDU header format

Parameter name	Format
command_code	USINT
Timeout	USINT
TransactionID	UINT

The command_code shall specify the type of UCMM_PDU as shown in Table 10. UCMM packets received with a reserved command_code shall be discarded without acknowledgement.

Table 10 – UCMM command codes

Command_code	Description
0	Reserved
1	acknowledge a request
2	request with retry until acknowledged
3	response with retry until acknowledged
4	request with no acknowledge and no response
5	acknowledge a response
6	response which shall not retry (no acknowledge)
7	request with retry until response (no acknowledge)
8	Request which shall not retry and shall cause a code 6 response
9 to 255	Reserved

The timeout field shall specify the duration of the transaction in an 8-bit floating-point format. The most significant 5 bits of the field shall be an unsigned exponent that is not biased. The least significant 3 bits shall be the least significant 3 bits of a 4 bit unsigned mantissa. The most significant bit of the mantissa shall be 1 and shall not appear explicitly in the 8-bit representation. The binary point shall be positioned between the implied 1 and the rest of the mantissa. The units of the transaction duration computed using the timeout field shall be in milliseconds.

NOTE Using ANSI C precedence rules, the number of 0,125 ms ticks is
(8 | timeout & 7) << (timeout >> 3 & 31).

The transactionID field shall be composed of two sub-fields:

- the least significant 10 bits, RECORD, shall identify a specific transaction;
- the most significant 6 bits, SEQUENCE, shall be used for duplicate detection. It shall be incremented for every unique message on this RECORD; however it shall not be incremented on a retry.

Only one message shall be outstanding for a given RECORD.

If multiple outstanding messages are required then multiple RECORDs are required.

When an I'm alive fixed tag packet is received from a node (see IEC 61158-4-2, 6.10 and 8.1), all UCMM transactions with that node shall be aborted.

4.1.4 Transport_Headers

4.1.4.1 General

Contents of Transport headers varies depending on the class of transport selected during the connection establishment.

4.1.4.2 Class 0 (null or base)

The class 0 transport header shall be an unsigned 16-bit number. This header shall be written to the TPDU buffer by the transport and shall be simply discarded by the transport from the packet coming from the consumer. The format of the Header is shown in Table 11.

Table 11 – Transport class 0 header

Parameter name	Format
don't_care	UINT

4.1.4.3 Class 1 (duplicate detection)

The class 1 transport header shall be an unsigned 16-bit sequence count. This header shall be written to the TPDU buffer by the transport and shall be read by the transport from the packet coming from the consumer. The format of the Header is shown in Table 12.

Table 12 – Transport class 1 header

Parameter name	Format
sequence_count	UINT

4.1.4.4 Class 2 (acknowledged)

Like the class 1 transport header, the class 2 transport header shall be a 16-bit sequence count. This header shall be written to the TPDU buffer by the transport and shall be read by the transport from the packet coming from the consumer. The format of the Header is shown in Table 13.

Table 13 – Transport class 2 header

Parameter name	Format
sequence_count	UINT

4.1.4.5 Class 3 (verified)

Like the class 1 transport header, the class 3 transport header shall be a 16-bit sequence count. This header shall be written to the TPDU buffer by the transport and shall be read by the transport from the packet coming from the consumer. The format of the Header is shown in Table 14.

Table 14 – Transport class 3 header

Parameter name	Format
sequence_count	UINT

4.1.4.6 Class 4 (non-blocking), class 5 (non-blocking, fragmenting) and class 6 (multipoint connection, fragmenting)

NOTE Subclause contents from the previous edition have been obsoleted.

4.1.4.7 Listen only O⇒T data format

The O=>T connection shall use a specific heartbeat format (no 32-bit header, 0 octets of application data).

4.1.4.8 Input only O⇒T data format

The O=>T connection shall use a specific heartbeat format (no 32-bit header, 0 octets of application data).

4.1.4.9 Exclusive owner O⇒T data format

Exclusive owner connections shall have either of two real-time transfer formats:

- 32-bit header, fixed size;
- no header, variable size.

The 32-bit header prefixed to the real-time data shall be in the form shown in Table 15.

Table 15 – Real-time data header – exclusive owner

Parameter	Format	Size (bits)
Run_idle	RUN_IDLE	1
Reserved	UDINT	31

The `run_idle` flag (bit 0) shall be set (1 = `RUN`) to indicate that the following data shall be sent to the target application. It shall be clear (0 = `IDLE`) to indicate that the idle event shall be sent to the target application. The `reserved` field (bits 1 to 31) shall be reserved and set to 0.

If the no_header transfer format is used, the reception of a packet with data beyond the transport header shall indicate the RUN mode. If the packet is truncated after the transport header, the target shall be sent an idle event.

4.1.4.10 Redundant owner O⇒T data format

4.1.4.10.1 General format

Redundant owner shall have a 32-bit header prefixed to the real-time data. This header shall be in the form shown in Table 16.

Table 16 – Real-time data header– redundant owner

Parameter	Format	Size (bits)
Run_idle	RUN_IDLE	1
COO	BOOLEAN	1
ROO	USINT	2
Reserved	UDINT	28

4.1.4.10.2 Run_idle flag

The run_idle flag (bit 0) shall be have the same meaning as it does in an exclusive owner connection and shall be one of RUN or IDLE. The reserved field (bits 4 to 31) shall be reserved and set to 0.

4.1.4.10.3 Claim output ownership (COO) flag

The COO flag shall be set (1) when an originator application wants its connection to be the owning connection of the target application. The COO flag shall be reset (0) when an originator application does not want its connection to be the owning connection of the target application. When the owning connection resets (0) its COO flag, its sibling connections shall be checked for a set (1) COO flag. The new owner shall be any of the connections that have their COO flag set.

NOTE This results in undefined behaviour if more than one other connection has its COO flag set.

4.1.4.10.4 Ready for ownership of outputs (ROO) priority value

The ROO priority value shall be non-zero when an originator application does not want to force its connection to be the owning connection of the target application, but is ready to be the owning connection should there be no originator applications claiming to be the owning connection. The ROO priority value shall be zero when the originator application does not want to be the owning connection of the target connection and is not to be the owning connection should there be no originator application claiming to be the owning connection. The ROO priority value shall be used only when the COO flag is reset.

The value of the ROO field can range from 0 to 3. The originator applications shall each determine a unique non-zero ROO value.

4.1.4.10.5 Determining the owning connection

The originator applications shall determine among themselves which originator application has the owning connection. The owning connection shall be determined by the originator application that sets its COO flag. In situations where multiple originator applications have their COO flag set or where no originator connections have their COO flag set, the following rules shall be applied by the target transport to determine the owning connection.

- There shall be no owning connection until an originator application sends a real-time packet with the COO flag set;
- If there is only one originator application which had the COO flag set in its last real-time packet, that originator application shall have the owning connection;
- If there are multiple originator applications which had the COO flag set in its last real-time packet, the last originator application that transitioned its COO flag from reset to set shall have the owning connection.
- If the originator application with the owning connection resets its COO flag, closes its connection, or if that connection times out, and no other originator applications have their COO flags set, the originator application with the highest non-zero ROO priority value shall have the owning connection.
- If all of the originator applications have their COO flags reset and ROO priority values set to zero, there shall be no owning connection.
- When the first real-time packet containing a set COO flag is received by the target transport, the originator application that sent the real-time packet shall have the owning connection.

4.1.4.10.6 Transporting events and data to a target application

The connection related events from each of the redundant owner connections shall be combined using the following rules such that the target application sees only a single exclusive owner connection:

- Until an owning connection is initially determined, the transport shall not indicate real-time data reception to the target application;
- If an owning connection is determined, the transport shall indicate the event consistent with the owning connections real-time data to the target application. If the Run/Idle flag is

reset in the real-time data for the owning connection, the transport shall indicate the idle state to the target application. If the Run/Idle flag is set in the real-time data for the owning connection, the transport shall indicate the run state and the real-time data to the target application;

- If an owning connection had been previously determined, but no originator is currently claiming or ready for ownership, the transport shall indicate idle state to the target application;
- If all the redundant connections are closed or have been timed out, the target transport shall indicate the event consistent with the last connection to have been closed or timed out to the target application.

4.1.5 CM_PDUs

4.1.5.1 General

Connection establishment and maintenance services are provided by the Connection Manager. They are

CM_Forward_Open (corresponds to Forward_Open and Large_Forward_Open PDUs)

CM_Forward_Close (corresponds to Forward_Close PDUs)

CM_Unconnected_Send (corresponds to Unconnected_Send PDUs)

CM_Get_Connection_Data (corresponds to Get_Connection_Data PDUs)

CM_Search_Connection_Data (corresponds to Search_Connection_Data PDUs)

CM_Get_Connection_Owner (corresponds to Get_Object_ Owner PDUs)

4.1.5.2 Connection Manager

The Connection Manager object is used to manage the establishment and maintenance of communication connections.

The Connection Manager objects at different nodes shall communicate using the UCMM services described in IEC 61158-5-2. The TPDUs with which peer Connection Managers communicate shall be derived from the services of the Connection Manager.

4.1.5.3 Forward_Open

4.1.5.3.1 General

The Forward_Open service is used to establish a connection with a target device. This service results in local connection establishment on each link along the path.

NOTE The processing of a single Forward_Open service can result in the creation of multiple active Connection object instances.

A Forward Open can be either a non-null Forward_Open or a null Forward_Open. A non-null Forward_Open is a Forward_Open service request for which at least one of the connection types in the O2T_ or T2O_connection_parameters field is not 00 (NULL). A null Forward_Open is a Forward_Open service request for which the connection type in both the O2T_ and T2O_connection_parameters fields is 00 (NULL) and results in no connection being established.

A Forward_Open (both null and non-null) can be either not matching or matching. A matching Forward_Open service request received by the target device is one where the Connection Triad matches an existing connection.

The usage of each of the combinations of non-null/null and not matching/matching Forward_Open is described in 4.1.5.3.2 and summarized in the list below:

- non-null / not matching – open a connection;
- non-null / matching – error;
- null / not matching – ping a device, or configure;
- null / matching – reconfigure.

The response to the Forward_Open_Request_PDU is a Forward_Open_ResponsePDU. The Forward_Open response shall contain the originator connection serial number and owner vendor and serial number and the connection IDs (CID) necessary to complete the connection if successful and error information if the connection failed.

4.1.5.3.2 Forward_Open variants

4.1.5.3.2.1 Non null Forward_Open, not matching

The Forward_Open request sets up network, transport, and application connections. An application connection consists of a single transport connection, and one or two network connections that are in turn comprised of multiple link connections. Each port segment in the connection path uses a link connection. The Forward_Open service between two devices builds one or two link connections as specified by the network connection parameter and the requested packet intervals (CM_RPI). Since up to two network connections can be required for a single transport connection, they are differentiated by the O2T and T2O designations; O2T means originator to target, and T2O means target to originator.

The path specified in the MR_Request_Header for the Forward_Open service shall not contain any Electronic Key segment. Electronic key segments may be included in the path specified in the connection_path parameter of the Forward_Open service.

Success shall be returned when the connection requested has been established from this point forward in the path. This response also shall indicate the connection serial number and the actual packet interval of the connection. Once the successful response has been received, the connection shall be open from this point forward in the path. Targets shall wait at least 10 seconds after sending the `CM_Forward_Open_Good_Response` for the first packet on a connection.

4.1.5.3.2.2 Non null Forward_Open, matching

A non-null Forward_Open for which the Connection Triad matches an existing connection shall return a general status = 0x01, extended status = 0x0100 (Connection in use or Duplicate Forward_Open).

The suggested originator behaviour in the non-null Forward_Open, matching case should be to either close and re-establish the connection or wait for the connection to time-out and then establish the connection again. It shall be recognized that in the latter case, it shall take the connection 60 s (first data time-out) to time-out before the connection can be re-established.

4.1.5.3.2.3 Null Forward_Open, not matching

A null Forward_Open for which the Connection Triad does not match an existing connection's parameters can be used for the following functions.

a) Ping a device, with the following characteristics:
   - the single application path "20 01 24 01" (i.e. Identity object);
   - an electronic key segment may be included;
   - no data segment is included;

- no connection is established.

b) Configure a device's application, with the following characteristics:

- a configuration application path and data segment shall be included in the request (the data is sent to the application specified by the path and applied);
- if the entire configuration cannot be applied, then none of the configuration shall be applied and the appropriate error code returned;
- an electronic key segment may be included;
- no connection is established.

A target device shall respond to a null Forward_Open request with one of the following errors when the target does not support the requested function ("ping a device" or "configure a device's application"):

- Null Forward_Open function not supported (general status 0x01, extended status 0x0132) (recommended).
- Invalid network connection parameter (general status 0x01, extended status 0x0108) (deprecated).

4.1.5.3.2.4 Null Forward_Open, matching

A Null Forward_Open for which the Connection Triad matches an existing connection's parameters can be used to reconfigure a target device's application. Intermediate nodes shall always forward these requests. The associated behaviours are:

- a configuration application path and data segment shall be included in the request and they are sent to the application to change the application configuration;
- if the entire configuration cannot be applied then none of the configuration shall be applied and the appropriate error code returned;
- the connection shall not be interrupted due to this request.

A target device shall respond to a null Forward_Open request with one of the following errors when the target does not support the requested function ("reconfigure a target device's application"):

- Null Forward_Open function not supported (general status 0x01, extended status 0x0132) (recommended).
- Invalid network connection parameter (general status 0x01, extended status 0x0108 (deprecated).

If the interpretation of the consumed/produced data changes as a result of the reconfigure operation, care shall be taken in the producing and consuming applications. There is no specified mechanism to coordinate between the producing and consuming applications, so a change in the meaning of the real time data can result in unexpected operation. Devices have the option to reject a reconfiguration request, with an Object state conflict error (general status = 0x0c), to prevent this situation.

4.1.5.3.3 Format of Forward_Open request

The format of the Forward_Open request TPDU shall be of the form shown in Table 17.

Table 17 – Forward_Open request format

Parameter name	Format
priority_and_tick	SWORD
connection time-out ticks	USINT
O2T_CID	UDINT

Parameter name	Format
T2O_CID	UDINT
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
connection timeout multiplier	USINT
reserved[3]	USINT
O2T_CM_RPI	UDINT
O2T_connection_parameters [a]	UINT
T2O_CM_RPI	UDINT
T2O_connection_parameters [a]	UINT
xport_type_and_trigger	SWORD
connection_path_size	USINT
connection_path	Padded EPATH
a Connection type in this parameter shall be 00 (NULL) for a null Forward_Open.	

4.1.5.3.4 Format of Forward_Open response if success

If the `status` parameter of the `CM_open_response` is zero (no error), the format of the Forward_Open response TPDU shall be of the form shown in Table 18.

Table 18 – Forward_Open_Good response format

Parameter name	Format
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
O2T_CM_API	UDINT
T2O_CM_API	UDINT
Application_reply_size; (in 16-bit words)	USINT
Reserved	USINT
application_reply[]	UINT

If a Connection object is instantiated to support this connection:

- the values of O2T_CM_API and O2T_CM_API are used for the Connection object Expected_packet_rate attribute, after converting to milliseconds.
- the Expected_packet_rate will not be used for the Connection object Inactivity/Watchdog Timer. See 4.1.6.2 for connection timeout handling.

4.1.5.3.5 Format of Forward_Open response if failure

If the `status` parameter of the `CM_open_response` is not zero (error), the format of the Forward_Open response TPDU shall be of the form shown in Table 19.

Table 19 – Forward_Open_Bad response format

Parameter name	Format
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
remaining_path_size; (in 16-bit words)	USINT
Reserved	USINT

This format shall be used for all failures of the connection system. The requested connection shall not be established, and the object specific status words shall contain information about the reason for the failure. The `remaining_path_size` shall contain the length of the path at the point the connection failed. This information can be used to debug the problem.

In the failure response, the remaining `remaining_path_size` shall be the "pre-stripped" size. This shall be the size of the path when the node first receives the request and has not yet started processing it. A target node may return either the "pre-stripped" size or 0 for the remaining `remaining_path_size`.

A duplicate Forward_Open service shall be defined as a Forward_Open service whose `originator_vendor_ID`, `connection_serial_number`, and `originator_serial_number` match an existing connection's parameters. If the duplicate Forward_Open service is a null Forward_Open service (defined as the connection type in both the O2T and T2O network connection parameter fields are NULL), then the Forward_Open service shall be forwarded to the application for further processing. Null Forward_Open requests may be used to reconfigure the connection. The Connection Manager in the intermediate nodes need not allocate additional resources for a duplicate Forward_Open request since the resources have already been allocated. If the duplicate Forward_Open request is not NULL, then an general status = 0x01, extended status = 0x0100 shall be returned.

4.1.5.4 Large_Forward_Open

4.1.5.4.1 General

The Large_Forward_Open shall have the same function and same behaviour as the Forward_Open except that it shall allow the establishment of connections larger than 511 octets.

The response to the Large_Forward_Open_Request_PDU is an Large_Forward_Open_ResponsePDU. All other requirements are identical to those specified for the Forward_Open service in 4.1.5.3.

4.1.5.4.2 Format of Large_Forward_Open request

The format of the Large_Forward_Open request TPDU shall be of the form shown in Table 20.

Table 20 – Large_Forward_Open request format

Parameter name	Format
priority_and_tick	SWORD
connection time-out ticks	USINT
O2T_CID	UDINT
T2O_CID	UDINT

Parameter name	Format
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
connection timeout multiplier	USINT
reserved[3]	USINT
O2T_CM_RPI	UDINT
O2T_ex_connection_size	UINT
O2T_connection_parameters	UDINT
T2O_CM_RPI	UDINT
T2O_ex_connection_size	UINT
T2O_connection_parameters	UDINT
xport_type_and_trigger	SWORD
connection_path_size	USINT
connection_path	Padded EPATH

The connection size field in the two connection_parameters shall be reserved in this case and set to zero. The ex_connection_size parameters shall be used instead to specify the maximum size of the connection (up to 65 535).

4.1.5.4.3 Format of Large_Forward_Open response if success

If the `status` parameter of the `CM_open_response` is zero (no error), the format of the Large_Forward_Open response TPDU shall be of the form shown in Table 21.

Table 21 – Large_Forward_Open_Good response format

Parameter name	Format
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
O2T_CM_API	UDINT
T2O_CM_API	UDINT
Application_reply_size; (in 16-bit words)	USINT
Reserved	USINT
application_reply[]	UINT

4.1.5.4.4 Format of Large_Forward_Open response if failure

If the `status` parameter of the `CM_open_response` is not zero (error), the format of the Large_Forward_Open response TPDU shall be of the form shown in Table 22.

Table 22 – Large_Forward_Open_Bad response format

Parameter name	Format
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
remaining_path_size; (in 16-bit words)	USINT
Reserved	USINT

Either a target device or an intermediate router shall return an Invalid Connection Size extended error code, along with the maximum connection size supported additional status word, if the connection size requested is not supported while processing a Large_Forward_Open request.

4.1.5.5 Forward_Close

4.1.5.5.1 General

The Forward_Close request shall remove a connection from all the nodes participating in the original connection. The Forward_Close shall be sent between Connection Managers as specified in the connection_path. The Forward_Close request shall cause all resources in all nodes participating in the connection to be deallocated, including connection IDs, link transmit time, and internal memory buffers.

If an intermediate node cannot find the connection that is to be closed (it may have timed out at the node), the Forward_Close request shall still be forwarded to downstream nodes or the target application (via the `CM_close_indication`).

NOTE The Forward_Close is always forwarded to allow downstream nodes or the target application to release any resources that were allocated for the connection.

4.1.5.5.2 Format of Forward_Close request

The format of the Forward_Close request shall be of the form shown in Table 23.

The path specified in the MR_Request_Header for the Forward_Close service shall not contain any Electronic Key segment. Electronic key segments may be included in the path specified in the connection_path parameter of the Forward_Close service.

Table 23 – Forward_Close request format

Parameter name	Format
connection_priority/tick time	SWORD
connection_timeout	USINT
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
connection_path_size	USINT
Reserved	USINT
connection_path	Padded EPATH

4.1.5.5.3 Format of Forward_Close response if success

The Forward_Close response shall be sent between Connection Managers in response to a Forward_Close request, and shall contain the status of the close request. If the `status` parameter of the `CM_close_response` is zero (no error), the format of the Forward_Close response TPDU shall be of the form shown in Table 24.

Table 24 – Forward_Close_Good response format

Parameter name	Format
connection serial number	UINT
originator_vendor ID	UINT
originator serial number	UDINT
application_reply_size; in 16-bit words	USINT
Reserved	USINT
application_reply[]	UINT

A successful Forward_Close response shall be returned when the close has been acknowledged by all the nodes from this point in the path to the end of the path. The response shall indicate the `connection_serial_number`, `originator_vendor_ID`, and `originator_serial_number` of the closed connection. Once the response indicating success has been received, the connection shall be closed from this point in the path to the end. The target application may optionally pass back application specific information in the successful close response. If no `application_reply` information is contained in the service the `application_reply_size` shall be zero.

4.1.5.5.4 Format of Forward_Close response if failure

If the `status` parameter of the `CM_close_response` is not zero (error), the format of the Forward_Close response shall be of the form shown in Table 25.

Table 25 – Forward_Close_Bad response format

Parameter name	Format
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
remaining_path_size	USINT
reserved	USINT

The object specific status words shall contain information about the reason for the failure. The `remaining_path_size` shall contain the path size from the point at which the close connection failed.

In the failure response, the `remaining_path_size` shall be the “pre-stripped” size. This shall be the size of the path when the node first receives the request and has not yet started processing it. A target node shall return either the “pre-stripped” size or 0 for the `remaining_path_size`.

4.1.5.6 Unconnected_Send

4.1.5.6.1 General

The Unconnected_Send service shall allow an application to send a message to a device without first setting up a connection. The Unconnected_Send service shall use the Connection Manager object in each intermediate node to forward the message and to remember the return path. The UCMM of each link shall be used to forward the request from Connection Manager to Connection Manager just as it is for the Forward_Open service; however. no connection shall be built. The Unconnected_Send service shall be sent to the local Connection Manager and shall be sent between intermediate nodes. When an intermediate node removes the last port segment, the embedded message request shall be formatted as an OM_Request and sent to the port and link address of the last port segment using the UCMM for that link type.

NOTE The target node never sees the Unconnected_Send service but only the embedded message request arriving via the UCMM.

4.1.5.6.2 Format of Unconnected_Send request

The format of the Unconnected_Send request shall be of the form shown in Table 26.

Table 26 – Unconnected_Send request format

Parameter name	Format	Description
priority/tick time	SWORD	Used to calculate request timeout information
conn time-out ticks	USINT	
Message_size	UINT	Number of octets in the embedded Message Router PDU
Message Router PDU	OM_Request	Embedded Messager Router PDU
pad	USINT	Only present if Message_size is an odd value. This allows the remaining fields to be aligned on a 16-bit boundary.
route_path_size	USINT	Number of 16-bit words in the route_path field
Reserved	USINT	Reserved, shall be set to zero (0)
route_path	Padded EPATH	Indicates the route to the remote target device

The path specified in the MR_Request_Header for the Unconnected_Send service shall not contain any Electronic Key segment. Electronic key segments may be included in the path specified in the MR_Request_Header of the embedded message request, and in the route_path parameter of the Unconnected_Send service.

The route path shall be of the form:

[Electronic Key segment1] } Port Segment Group for 1st intermediate node
Port segment1 }

[[Electronic Key segment2] } Port Segment Group for 2nd intermediate node
Port segment2] }

…

[[Electronic Key segmentN] } Port Segment Group for Nth intermediate node
Port segmentN] }

All segments listed in brackets [...] are optional. If the Electronic Key segment is present the target shall evaluate the key. See 4.1.9.3 for the definition of Port segment and 4.1.9.4.2 for the definition of Electronic Key segment.

4.1.5.6.3 Unconnected_Send response

The Unconnected_Send response shall be generated by the last intermediate node from the UCMM response generated by the target node or by an intermediate node as the result of a UCMM time-out, a problem with the embedded message, or a problem with the Unconnected Service Request itself. The packet shall be routed from intermediate node to intermediate node using the information stored when the Unconnected_Send request was processed. The response shall contain a header with status information about the request and a variable length response generated by the target node.

4.1.5.6.4 Format of Unconnected_Send response if success

This format shall be used when a successful Unconnected_Send response is received. The structure of the response is shown in Table 27.

Table 27 – Unconnected_Send_Good response format

Parameter name	Format
application_response [ ]	OCTET

The application_response field contains the data returned by the target device/object in the Service_ResponsePDU for the embedded request. If the Service_ResponsePDU returned by the target device/object did not contain any data, then this field shall be empty.

EXAMPLE This field would contain Attribute Data in response to an embedded Get_Attribute_Single request.

4.1.5.6.5 Format of Unconnected_Send response if failure

In case of a failure of the Unconnected_Send service, the Service code returned in the MR_Response_Header for the Unconnected_Send service may correspond either to the Service code sent inside the Unconnected_Send service data (embedded message request) or to the Unconnected_Send Service code itself:

- when a router detects the error, the Unconnected_Send response Service code (0xD2) shall be returned in the MR_Response_Header;
- when the target device detects the error, the response Service code for the embedded message request shall be returned in the MR_Response_Header.

EXAMPLE If the target device detects the error and the requested service was a Get_Attribute_Single, then the Service code in the MR_Response_Header will be 0x8E.

This is advantageous to originators and routers because:

- of the two service codes (Unconnected Send and embedded), the service code that failed is returned to the originator;
- routers are not required to parse the Service code of the embedded message within the Unconnected_Send request.

The format of the Unconnected_Send response service for a failure shall be of the form shown in Table 28.

Table 28 – Unconnected_Send_Bad response format

Parameter name	Format
remaining_path_size	USINT
pad	USINT

The remaining_path_size field is only present with routing type errors and indicates the number of words in the original route path (route_path parameter of the Unconnected Send request) as seen by the router that detects the error. The pad field is reserved and shall be set to zero.

4.1.5.6.6 CP 2/3 modifications

4.1.5.6.6.1 General

The services of the Connection Manager are not supported by CP 2/3 nodes when they are the target of the request. Thus, only nodes which provide routing to or from CP 2/3 support the Connection Manager object. When the target node of a message is on CP 2/3, these routers are required to translate the message into normal CP 2/3 explicit messaging format (see IEC 62026-3). As a result, CP 2/3 nodes require no changes to accept routed messages from other Type 2 subnets.

In order to allow these routers the ability to handle multiple outstanding transactions on CP 2/3, the Unconnected_Send service of the Connection Manager is modified when traversing a CP 2/3 subnet, as specified in 4.1.5.6.6.2 and 4.1.5.6.6.3..

4.1.5.6.6.2 Unconnected_Send request modification

When sent on a CP 2/3 subnet, the Unconnected_Send service data is prepended with a 16 bit transaction ID. This transaction identifier is generated by the requesting device and returned by the router along with the response from the target. The modified service request is as shown in Table 29.

Table 29 – Unconnected_Send request format (modified)

Parameter name	Format	Description
Transaction_ID	UINT	Used for transaction management in the CP 2/3 requesting device and CP 2/3 routers. This field is used for transaction matching by message originators on CP 2/3 not passed on to other subnets.
priority/tick time	SWORD	Used to calculate request timeout information
conn time-out ticks	USINT	
Message_size	UINT	Number of octets in the embedded Message Router PDU
Message Router PDU	OM_Request	Embedded Messager Router PDU
pad	USINT	Only present if Message_size is an odd value. This allows the remaining fields to be aligned on a 16-bit boundary.
route_path_size	USINT	Number of 16-bit words in the route_path field
Reserved	USINT	Reserved, shall be set to zero (0)
route_path	Padded EPATH	Indicates the route to the remote target device

4.1.5.6.6.3 Unconnected_Send response modification

A CP 2/3 router shall always return a successful response to an Unconnected_Send service request. This success response may actually indicate that an error was encountered. This is necessary because additional error information is needed to handle routing errors and this additional information does not conform to the Error Response format defined for CP 2/3 (see IEC 62026-3). As specified in 4.1.5.6.5, the Service code returned may be either the Service code sent inside the Unconnected_Send service data or the Unconnected_Send Service code. The 0x94 Error Response Service code (see IEC 62026-3) is never returned.

The modified successful and unsuccessful service responses are as shown in Table 30 and Table 31. This is the data placed within the application_ response service data area; the response Service code is earlier in the packet.

Table 30 – Unconnected_Send_Good response format (modified)

Parameter name	Format	Description
Transaction_ID	UINT	Echoes the value received in the associated Unconnected_Send request
General status	USINT	This value is zero (0) for successful transactions.
Reserved	USINT	Shall be zero (0).
actual_application_response [ ]	OCTET	

The actual_application_response field contains the data returned by the target device/object in the Service_ResponsePDU for the embedded request. If the Service_ResponsePDU returned by the target device/object did not contain any data, then this field shall be empty.

EXAMPLE This field would contain Attribute Data in response to an embedded Get_Attribute_Single request.

The response Service Data associated with an unsuccessful Unconnected_Send response is defined below.

Table 31 – Unconnected_Send_Bad response format (modified)

Parameter name	Format	Description
Transaction_ID	UINT	Echoes the value received in the associated Unconnected_Send request
General Status	USINT	One of the General Status codes listed in Table 180. If a routing error occurred, it shall be limited to the values indicated as detected by router devices in Table 179.
Size of Additional Status	USINT	Number of 16 bit words in the Additional Status array
Additional Status [ ]	UINT	When returning an error from a target which is a CP 2/3 node, the Additional Status shall contain the 8 bit Additional Error Code from the target in the lower 8 bits and a zero (0) in the upper 8 bits.
remaining_path_size	USINT	This field is only present with routing type errors and indicates the number of words in the original route path (route_path parameter of the Unconnected_Send request) as seen by the router that detects the error.

4.1.5.7 Get_Connection_Data

4.1.5.7.1 General

This service shall return the parameters associated with a specified connection number. The connection number may be different from device to device even for the same connection. The connection number corresponds to the offset into the Connection Manager attribute that enumerates the status of the connections.

NOTE This service can be used for network diagnostics.

4.1.5.7.2 Format of Get_Connection_Data request

The format of the Get_Connection_Data request shall be of the form shown in Table 32.

Table 32 – Get_Connection_Data request format

Parameter name	Format
connection_number	UINT

4.1.5.7.3 Format of Get_Connection_Data response

The format of the Get_Connection_Data response TPDU shall be of the form shown in Table 33.

Table 33 – Get_Connection_Data response format

Parameter name	Format
connection_number	UINT
connection_state	UINT
originator_port	UINT
target_port	UINT
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
originator_O2T_CID	UDINT
target_O2T _CID	UDINT
O2T_connection timeout multiplier	USINT
reserved1[3]	USINT
originator_O2T_CM_RPI	UDINT
originator_O2T_CM_API	UDINT
originator_T2O_CID	UDINT
target_ T2O_CID	UDINT
T2O_connection timeout multiplier	USINT
reserved2[3]	USINT
originator_T2O_CM_RPI	UDINT
originator_T2O_CM_API	UDINT

4.1.5.8 Search_Connection_Data

4.1.5.8.1 General

This service shall return the parameters associated with a specified connection within a device identified by vendor and serial number.

NOTE This service can be used for network diagnostics.

4.1.5.8.2 Format of Search _Connection_Data request

The format of the Search_Connection_Data request shall be of the form shown in Table 34.

Table 34 – Search_Connection_Data request format

Parameter name	Format
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT

4.1.5.8.3 Format of Search _Connection_Data response

The format of the Search_Connection_Data response shall be the same as the response from the Get_Connection_Data service (see 4.1.5.7).

4.1.5.9 Get_Connection_Owner

4.1.5.9.1 General

The Get_Connection_Owner service of the Connection Manager shall return data about the connection(s) that own(s) a particular object. It shall be implemented in any device that accepts redundant connections.

4.1.5.9.2 Format of Get_Connection_Owner request

The format of the Get_Connection_Owner request TPDU shall be of the form shown in Table 35:

Table 35 – Get_Connection_Owner request format

Parameter name	Format
reserved	USINT
path_size	USINT
path[]	Padded EPATH

The reserved field shall be set to zero.

4.1.5.9.3 Format of Get_Connection_Owner response

The format of the Get_Connection_Owner response TPDU shall be of the form shown in Table 36.

Table 36 – Get_Connection_Owner response format

Parameter name	Format
number_of_connections	USINT
number_claiming_ownership	USINT
number_ready_for_ownership	USINT
last_action	USINT
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT

4.1.6 CM PDU components

4.1.6.1 Network connection parameters

4.1.6.1.1 Format

Network connection parameters shall be provided as a single 16-bit word that contains five fields, shown in Figure 4. The fields within the 16-bit word shall indicate

- the type of network connection desired;
- whether the buffer size is variable or fixed;
- the priority of the connection;
- the size of the connection buffer required.

The size of the connection buffer shall include any transport header.

Network Connection Parameters

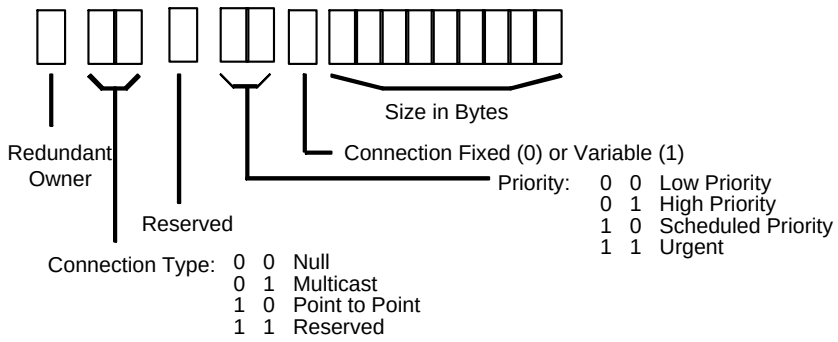


Figure 4 – Network connection parameters

4.1.6.1.2 Connection type

The connection type shall be one of NULL, MULTIPOINT, or POINT2POINT. The NULL type shall indicate that no network connection is required, while MULTIPOINT and POINT2POINT refer to the network connection types as defined in IEC 61158-5-2. The NULL connection type may be used to reconfigure a connection.

4.1.6.1.3 Priority

Priority shall be one of `LOW`, `HIGH`, `SCHEDULED` and `URGENT`, with `URGENT` being the highest priority (see 4.1.6.3 for detailed specification and usage).

For CP 2/1, `SCHEDULED` data is prioritized on a link wide basis; however, the two unscheduled priorities (`LOW` and `HIGH`) are arbitrated independently at each node. One node may transmit a `LOW` priority packet while another node has `HIGH` priority data to send.

4.1.6.1.4 Connection fixed/variable

The connection can be set up to use fixed or variable sized connections. With a fixed size connection, each transmission on the connection shall use the same size buffer. Otherwise, either a buffer overrun or underrun condition shall occur. With a variable sized connection, each transmission on the connection may send a variable amount of data up to the maximum size, which shall be specified when the connection is opened. Buffer underruns shall not be reported, but a buffer overrun can still occur if too much data is sent.

4.1.6.1.5 Connection size

The network connection size shall be the size of the buffer required for the connection, which includes the data and any transport header. For a variable sized connection, the size shall be the maximum size of the buffer for any transfer. The actual size of the transfer for a variable connection shall be equal to or less than the size specified for the network connection. The maximum buffer size shall be dependent on the links that the connection traverses.

4.1.6.1.6 Redundant owner

The redundant owner bit in the O⇒T direction shall be set (= 1) to indicate that more than one owner may be permitted to make a connection simultaneously. The bit shall be clear (= 0) to indicate an exclusive-owner, input only or listen-only connection.

4.1.6.1.7 Reserved parameters

Reserved bits shall be clear (= 0).

4.1.6.2 Requested and actual packet interval (CM_RPI and CM_API)

The connection originator provides a requested time between productions for each direction of the connection. The target provides the actual time between productions. When a connection traverses CP 2/1, the actual time may be further adjusted as provided in 4.1.6.3.

The requested packet interval (CM_RPI) shall be the requested time between packets in microseconds. The format of the CM_RPI shall be a 32-bit integer in microseconds.

The actual packet interval (CM_API) shall be the maximum time between packets in microseconds. The API is equivalent to the Transmission Trigger Timer value specified in IEC 61158-5-2, 6.2.3.2.1.7. The format of the CM_API shall be a 32-bit integer in microseconds.

The influence of the API and RPI values on the behavior of the connection is dependent on the Transport Class and Trigger definitions in 4.1.8.9.1.2. The CM_RPI value shall be used to allocate capacity usage at each of the producing nodes. The allocation of capacity usage may have to be adjusted when the CM_API is returned, since it is possible for the CM_RPI and CM_API to differ.

The CM_API shall equal the CM_RPI within the device's timer resolution limits, except as indicated for CP 2/1 in 4.1.6.3. If the CM_API is not equal to the CM_RPI it shall be faster than the CM_RPI unless the resulting CM_API would be 0, in which case the CM_RPI cannot be supported and an error shall be returned, either:

- CM_RPI Value(s) Not Acceptable, General Status 0x01, Extended Status 0x0112 (recommended);
- CM_RPI Not Supported, General Status 0x01, Extended Status 0x0111.

The Inactivity/Watchdog time-out value at each of the intermediate and target nodes shall be set to the connection time-out multiplier times the CM_RPI. If the resulting time-out is not achievable within the device's timer resolution limits, the time-out shall be rounded up the next value within the device's timer resolution limits.

4.1.6.3 Function of priorities

The order of priorities, from highest to lowest are: urgent, scheduled, high and low.

While there are no specific implementation requirements for end devices to internally differentiate traffic with different priorities, devices are recommended to give preferential processing to higher priority connections over lower priority connections. At a minimum,

devices are recommended to give higher priority to processing implicit messages over explicit messages.

The mapping of CIP Priorities to link priorities is specific to each CP.

Urgent priority

Urgent priority shall only be used where defined explicitly in the Type 2 companion standards.

EXAMPLE Urgent priority is used for I/O connections established with the Motion Device Axis object specified in IEC 61800-7-202.

On CP 2/1:

- bandwidth shall be reserved using the Network Schedule segment;
- the data interval shall be restricted to the CM_API, which means that if data arrives at an intermediate node faster than the API, the node shall filter the packets to the CM_API on CP 2/1;
- a tool that performs scheduling operations shall favor urgent priority over scheduled priority connections.

On other Type 2 links:

- nodes shall map the urgent priority to the corresponding network specific link priority.

Scheduled Priority

On CP 2/1:

- bandwidth shall be reserved using the Network Schedule segment;
- the data interval shall be restricted to the CM_API, which means that if data arrives at an intermediate node faster than the CM_API, the node shall filter the packets to the CM_API on CP 2/1;
- a tool that performs scheduling operations shall favor urgent priority over scheduled priority connections.

On other Type 2 links:

- nodes shall map the scheduled priority to the corresponding network specific link priority.

High Priority

Nodes shall map the high priority to the corresponding network specific link priority.

Low Priority

Nodes shall map the low priority to the corresponding network specific link priority.

4.1.6.4 Connection time-out multiplier

The Connection Time-out Multiplier specifies the multiplier applied to the CM_RPI to obtain the connection time-out value. Device shall stop transmitting on a connection whenever the connection times out even if the pending close has been sent. The multiplier shall be as represented by Table 37.

Table 37 – Time-out multiplier

Value	Multiplier	Notes
0	X 4	Default
1	X 8	
2	X 16	
3	X 32	
4	X 64	
5	X 128	
6	X 256	
7	X 512	
8 to 255	reserved	

4.1.6.5 Connection request priority and tick time

Two values are packed into this field. The connection request priority determines the priority of the UCMM messages used to establish the connection and has no correlation with the priority of the connection being made. Although Low priority is used for most connection establishments, High priority may be used for those special circumstances which require a connection be made quickly. The use of low priority was chosen to avoid congestion with time-critical messaging. UCMM messages can not use scheduled priority.

The tick time shall be used in conjunction with the connection request time-out ticks value. This value determines the time between ”ticks” in the latter field. Thus, if the tick time is 1ms., then a connection request time-out value of 5 would translate into 5 ms. If the tick time was 2 ms, then a connection request time-out value of 5 would translate into 10 ms.

The format of the Connection Request Priority/Tick Time shall be as shown in Figure 5.

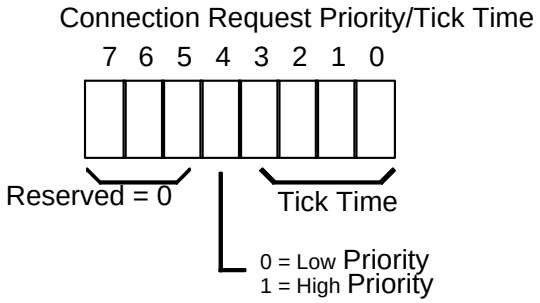


Figure 5 – Time tick

Table 38 shall determine the time between ticks.

Table 38 – Time tick units

Tick time		
Tick time (binary)	**Time per tick**	**Max. time**
0000	1 ms	255 ms
0001	2 ms	510 ms
0010	4 ms	1 020 ms
0011	8 ms	2 040 ms
0100	16 ms	4 080 ms

Tick time		
Tick time (binary)	**Time per tick**	**Max. time**
0101	32 ms	8 160 ms
0110	64 ms	16 320 ms
0111	128 ms	32 640 ms
1000	256 ms	65 280 ms
1001	512 ms	130 560 ms
1010	1 024 ms	261 120 ms
1011	2 048 ms	522 240 ms
1100	4 096 ms	1 044 480 ms
1101	8 192 ms	2 088 960 ms
1110	16 384 ms	4 177 920 ms
1111	32 768 ms	8 355 840 ms

4.1.6.6 Connection request time-out ticks

The connection request time-out ticks shall be used to specify the amount of time the originating application shall wait for the connection to be established. Each hop of the connection has a link specific time-out value implemented by the UCMM, and after the automatic retries have been exhausted without an acknowledge the UCMM shall generate a time-out error. Therefore if a device in the path shall be either not present or too busy to acknowledge a UCMM request, a UCMM time-out shall occur as fast as would be possible. The only time the connection request time-out period would be encountered shall be if the target node acknowledges the connection request but fails to respond or if the connection request shall be delivered but a failure in the network prevents the response from being returned. The connection request time-out shall be not the path time-out value derived from the CM_RPI and used to close a connection after it shall be established. The connection request time-out shall be specified by a combination of the connection request time-out ticks multiplied by the time per tick which shall be determined by the tick time value.

Since a particular connection request time-out can be expressed in several different formats, the rule shall be to **always pack it such that the tick time value shall be minimized.** This allows the best granularity for decreasing the time-out as the message is passed through hops. This also means that the tick time value may change as the message is passed through multiple hops. As an example, 300 ms may be encoded as:

1. 0001 10010110 -> Time per Tick = 2 ms, Ticks = 150 (right)

2. 0010 01001011 -> Time per Tick = 4 ms, Ticks = 75 (wrong)

The connection request time-out value shall be dependent on the number of port segments in the connection path, the processing time at the target node, and the congestion of the intermediate networks. The connection request time-out can be a relatively large value since the full connection request time-out shall only occur in a few rare cases. The only time the full connection request time-out would occur would be if:

- the connection request was successfully delivered to the target node and the node failed to generate a response. This would usually be caused by a firmware error in the device.
- the connection request was successfully delivered to the target node and the response was discarded due to congestion at an intermediate node.
- the connection request was successfully delivered to the target node and the target node or one of the intermediate nodes was removed or powered down before the response was returned.

The UCMM shall be used to forward the connection request to the next node in the path. The UCMM sends the request and then waits a link specific time for an acknowledgement or response, if no acknowledgement or response shall be received before the link specific time-out, the request shall be retried. After a link specific number of retries (typically three), the UCMM shall signal a time-out on the connection request. The Connection Manager which generated the request shall return an error response to the connection request. Therefore the connection request shall time-out as soon as possible for the most common system faults

- missing or powered down devices in the path;
- congestion at a device in the path.

To aid in debugging systems, each hop in the path shall decrement the time-out by about 80 ms before sending it out in the Forward_Open to the next device in the path. Each intermediate device (router) shall round the connection time-out value that shall be decremented such that a minimum of 1 tick shall be subtracted. This shall enable the last device before the failure to time-out first and return information about how far the connection was made before the failure. This information can be used to help in debugging problems in a system. A typical system function is shown in Figure 6.

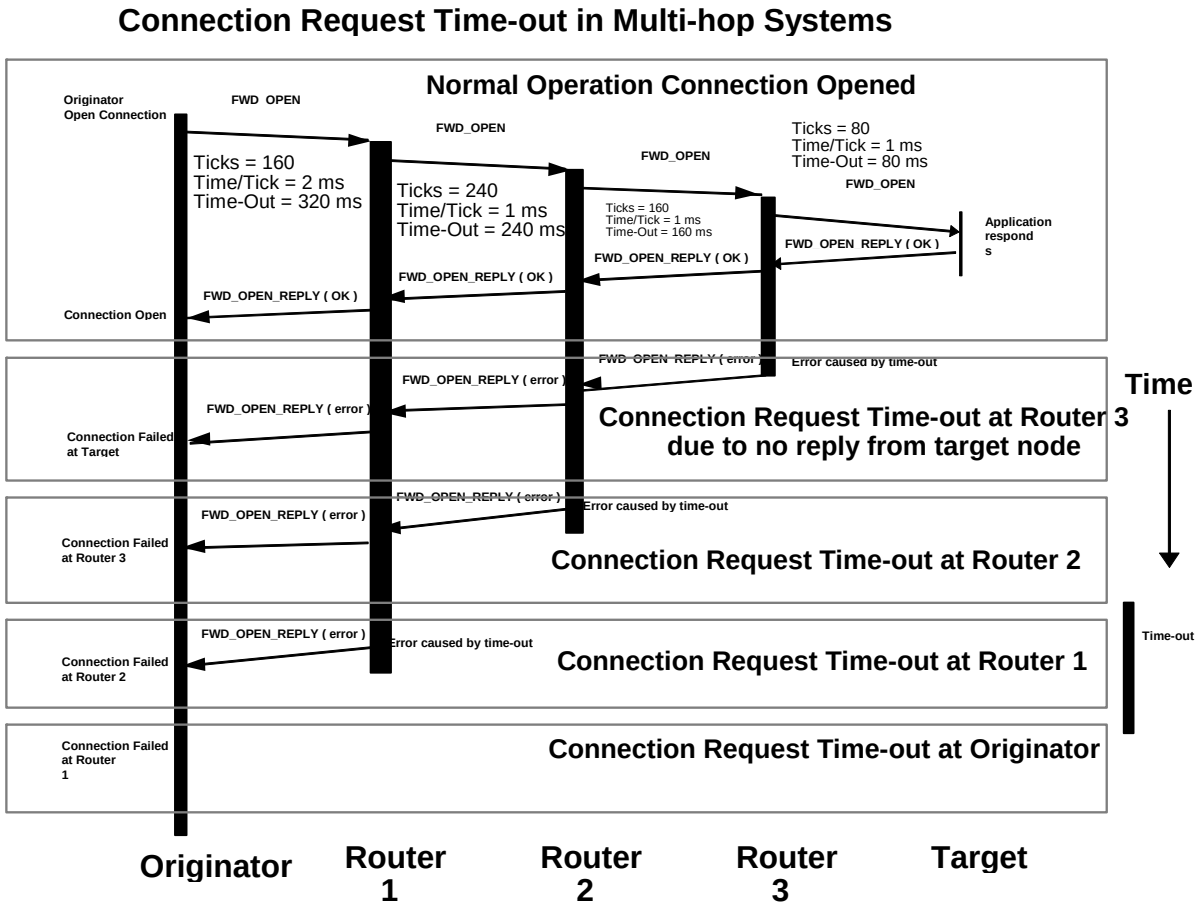


Figure 6 – Connection establishment time-out

In Figure 6 a connection is requested over a path containing 4 hops. The Connection request Time-out is specified as 320 ms by the originator of the connection. In the first case the connection is completed and the calculated time-out values are shown at each hop in the path. The second case is where the last device receives an acknowledgement but does not receive a response and times out. It returns a time-out error to the originator. Since the time-out value decreases for each hop, the time-out error shall be returned to the originator before the originator times out.

The other cases show time-outs at the other devices, and in each case the time-out error shall be returned to the originator before the originator itself times out. The last case is where no response is received by the originator and the connection open process is timed out. The reason to decrease the time-out at each node is so additional error information indicating where in the path the time-out occurred can be sent. Remember that if any of the devices do not receive an acknowledgement, the connection request shall immediately be timed out and an error response shall be generated.

4.1.6.7 Connection serial number

The connection serial number shall be a unique 16-bit value selected by the Connection Manager at the originator of the connection. The originator shall make sure that the 16-bit value is unique for the device. There shall be no other significance placed on the number by any other nodes in the connection path. The connection serial numbers shall be unique but do not have to be sequential. For example, an operator interface may have a large number of connections open at the same time, each with a unique number. The same values could be repeated at other operator interface stations. A possible implementation would be to have a connection list which points to the descriptor for each connection, and the connection serial number could be the index into the table.

4.1.6.8 Vendor ID

The vendor number shall be a unique number assigned to the various vendors of products. Each vendor has a unique number assigned.

NOTE Vendor IDs are assigned by ODVA, Inc.

4.1.6.9 Originator serial number

The originator serial number shall be a unique 32-bit value that is assigned to a device at the time of manufacture. This value shall be guaranteed to be unique for all devices manufactured by the same vendor. No significance shall be attached to the number. The combination of Vendor ID and Originator Serial Number shall be unique throughout the entire system.

4.1.6.10 Connection number

The connection number shall be a 16-bit value that is assigned by the Connection Manager when a connection is opened. This value allows other nodes to obtain connection data from the Connection Manager. This number shall not be confused with the Connection Serial Number.

4.1.6.11 Connection path size

The connection path size shall be the length of the connection path in 16-bit words. The length of the connection path varies during the connection process, since each node in the connection path removes the current port segment and forwards only the remaining path segments to the next node.

4.1.6.12 Connection path /connection remaining path

The connection path parameter shall contain one or more encoded paths as required by a combination of the service and the value of other parameters within the service data. Routing and related information (such as port, network, and electronic key segments), if present, shall be prior to the application path(s), as shown below.

The connection path shall be of the form:

[[Electronic Key segment1]	}	
[Network segment11]...[Network segment1N]	}	Port Segment Group for 1st intermediate node
Port segment1]	}	
[[Electronic Key segment2]	}	
[Network segment21]...[Network segment2N]	}	Port Segment Group for 2nd intermediate node
Port segment2]	}	
...		
[[Electronic Key segmentM]	}	
[Network segmentM1]...[Network segmentMN]	}	Port Segment Group for Mth intermediate node
Port segmentM]	}	
[Electronic Key segmentP]	}	
[Network segmentP1]...[Network segmentPN]	}	
Application Path1	}	Application Segment Group for target
[Application Path2]	}	
[Application Path3]	}	
[Simple Data segment]		

All segments listed in brackets [...] are optional. The Connection Manager object of both intermediate and target nodes shall process all segments that are directed to it.

Network segments are allowed at the specified locations only if allowed by the specific network segment definition.

When a configuration data segment is included in the connection path the configuration data segment shall follow the applications path(s).

The components of a connection path are specified in 4.1.9.1 to 4.1.9.7, and the hierarchy of a connection path is specified in 4.1.9.8.

Depending on the O2T_connection_parameters and T2O_connection_parameters fields and the presence of a data segment, one or more encoded application paths shall be specified. In general, the application paths are in the order of configuration path, consumption path, and production path. However, a single encoded path can be used when configuration, consumption, and/or production use the same path. See Table 39 for the valid combinations and the implied meaning of the application paths. The application paths are relative to the target node.

Table 39 – Encoded application path ordering

Network connection parameters		Data segment present	Number of encoded application paths		
O2T connection type	T2O connection type		1	2	3
NULL	NULL	Yes	Path is for configuration	First path is for configuration, second path is ignored	First path is for configuration, second and third paths are ignored
		No	Path is for pinging a device [a]	Invalid	Invalid
not NULL	NULL	Yes	Path is for configuration and consumption	First path is for configuration, second path is for consumption	Invalid
		No	Path is for consumption	Invalid	Invalid
NULL	not NULL	Yes	Path is for configuration and production	First path is for configuration, second path is for production	Invalid
		No	Path is for production	Invalid	Invalid
not NULL	not NULL	Yes	Path is for configuration, consumption and production	First path is for configuration, second path is for consumption and production	First path is for configuration, second path is for consumption, third path is for production
		No	Path is for consumption and production	First path is for consumption, second path is for production	First path is ignored, second path is for consumption, third path is for production

[a] If path is “20 01 24 01” used to ping a device, see 4.1.5.3.2.3 (null Forward_Open, not matching), invalid otherwise.

Encoded path compression rules are specified in 4.1.9.9.

4.1.6.13 Network connection ID

The Network Connection ID shall be link specific and shall not be related to the connection serial number, which is connection specific and the same over all the links. The fields of the Network Connection ID shall be used to set the screening mechanism for the specified link.

The Network connection ID is either a produced connection ID or a consumed connection ID.

A multicast connection ID shall not be reused until all connections associated with the connection ID have been closed or timed out.

The network connection ID is further specified in the DMPMs corresponding to the different link layers which can be associated with the Type 2 application layer (see Clause 10, Clause 11 and Clause 12).

4.1.6.14 Transport class and trigger

The transport class and trigger specify the type of transport and the production trigger required for the connection. The octet that encode the transport class and trigger shall be of the following form shown in Table 40.

Table 40 – Transport class, trigger and Is_Server format

TransportClass (4 bits)	TriggerMode (3 bits)	Is_Server (1 bit)
NULL = 0 DUPLICATE_DETECT = 1 ACKNOWLEDGED = 2 VERIFIED = 3 NONBLOCKING = 4 FRAGMENTING = 5 MULTIPOINT_FRAG = 6	CYCLIC = 0 CHANGE_OF_STATE = 1 APPLICATION = 2	IS_NOT_SERVER = 0 IS_SERVER = 1

The least significant 4 bits, `class`, shall determine which transport type is specified and shall be in the range 0 to 6. The `class` values 7 through 15 shall be reserved. Bits 4 to 6, called `trigger`, shall determine what event causes the production of data on the connection and shall be in the range 0 to 2. The `trigger` values 3 through 7 shall be reserved. For transports that need to specify the target as a server, the `is_server` field shall be 1; otherwise, it shall be 0. For transport classes 0 and 1, the `is_server` field shall be ignored. Bit 7 shall be the `is_server` field;

See IEC 61158-5-2 for details on the behaviour for the different combinations of `is_server`, `class`, and `trigger`, CM_RPI, and CM_API.

4.1.7 MR headers

4.1.7.1 MR request packets

All request packets delivered to the Message Router, either from a connection or from the UCMM, shall have a header of the form shown in Table 41.

Table 41 – MR_Request_Header format

Parameter	Format
Service	USINT
path_size	USINT
path	Padded EPATH

The first octet, service, shall be directly delivered to the application object and shall be in the range 1 through 127. The `path_size` shall determine the number of 16-bit words in the `path` field.

The `path` field shall contain addressing information for the packet (application path) and other information, and shall be padded.

The `path` field shall be of the form:

[Electronic Key segment]

Application Path

The Electronic Key segment is conditional. If the request is being sent across a connection the Electronic Key segment is not allowed, otherwise it is optional. If the Electronic Key segment is present the target shall evaluate the key. See 4.1.9.4.2 for the definition of Electronic Key segment and see 4.1.9.8 for the definition of the Application Path format.

EXAMPLE

34 04 42 42 0C 00 01 00 81 01	Electronic Key segment indicating the target device shall be compatible with a Vendor 0x4242, Device Type 0x000C, Product Code 0x0001, Major Revision 0x01, Minor Revision 0x01 device
20 04 24 01 30 03	Application path that specifies Instance 1 of the Assembly Object, Attribute 3.

4.1.7.2 MR response packets

Response packets from the Message Router shall have a header of the form shown in Table 42.

Table 42 – MR_Response_Header format

Parameter	Format
Service	USINT
Reserved	USINT
general_status	USINT
Extended_status_size	USINT
Extended_status[]	WORD

The `service` field shall be the value of the `MR_Request_Header.service` field with its most significant bit set (i.e. plus 0x80). If `MR_Request_Header.service` equals 0x05, the `MR_Response_Header.service` shall equal 0x85. The `general_status` field shall be filled using the `Status Code` parameter of the service response. Likewise the `extended_status`, `extended_status_size`, and `response` shall be filled using the `Extended Status` and other response parameters from the service response, respectively. If `extended_status_size` is 0, there is no `extended_status`. The `response_size` shall not be explicitly included in the `MR_Response_Header`; it shall be implied by number of octets in the `MR_Response_Header`.

4.1.8 OM_Service_PDU

4.1.8.1 General syntax

4.1.8.1.1 Get_Attribute_All

The Get_Attribute_All_RequestPDU body is empty.

The Get_Attribute_All_ResponsePDU body shall be as specified in Table 43.

Table 43 – Structure of Get_Attribute_All_ResponsePDU body

Name	Data type
Attribute Data	Object/class attribute-specific

4.1.8.1.2 Set_Attribute_All

The Set_Attribute_All_RequestPDU body shall be as specified in Table 44.

Table 44 – Structure of Set_Attribute_All_RequestPDU body

Name	Data type
Attribute Data	Object/class attribute-specific

The Get_Attribute_All_ResponsePDU body is empty.

4.1.8.1.3 Get_Attribute_List

The Get_Attribute_List_RequestPDU body shall be as specified in Table 45.

Table 45 – Structure of Get_Attribute_List_RequestPDU body

Name	Data type
Attribute Count	UINT
Attribute List	Array of UINT's

The Get_Attribute_List_ResponsePDU body shall be as specified in Table 46.

Table 46 – Structure of Get_Attribute_List_ResponsePDU body

Name	Data type	Semantics of values
Attribute Count	UINT	
Attribute Data	Array of STRUCT	
Attribute Identifier	UINT	
Attribute Status	UINT	Values 0 to 255 are reserved to mirror common service status codes. Values 256 to 65 535 are available for object/class attribute-specific errors.
Attribute Value	Object/class attribute- specific	

4.1.8.1.4 Set_Attribute_List

The Set_Attribute_List_RequestPDU body shall be as specified in Table 47.

Table 47 – Structure of Set_Attribute_List_RequestPDU body

Name	Data type
Attribute Count	UINT
Attribute Data	Array of STRUCT
Attribute Identifier	UINT
Attribute Value	Object/class attribute- specific

The Set_Attribute_List_ResponsePDU body shall be as specified in Table 48.

Table 48 – Structure of Set_Attribute_List_ResponsePDU body

Name	Data type	Semantics of values
Attribute Count	UINT	
Attribute Status List	Array of STRUCT	
Attribute Identifier	UINT	

Name	Data type	Semantics of values
Attribute Status	UINT	Values 0 to 255 are reserved to mirror common service status codes. Values 256 to 65 535 are available for object/class attribute-specific errors

4.1.8.1.5 Reset

The Reset_RequestPDU body shall be as specified in Table 49, if the **optional** "Object Specific Data" service parameter of the Reset request is specified. Else it shall be empty.

Table 49 – Structure of Reset_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Reset_ResponsePDU body shall be as specified in Table 50, if the **optional** "Object Specific Data" service parameter of the Reset response is specified. Else it shall be empty.

Table 50 – Structure of Reset_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

4.1.8.1.6 Start

The Start_RequestPDU body shall be as specified in Table 51, if the **optional** "Object Specific Data" service parameter of the Start request is specified. Else it shall be empty.

Table 51 – Structure of Start_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Start_ResponsePDU body shall be as specified in Table 52, if the **optional** "Object Specific Data" service parameter of the Start response is specified. Else it shall be empty.

Table 52 – Structure of Start_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

4.1.8.1.7 Stop

The Stop_RequestPDU body shall be as specified in Table 53, if the **optional** "Object Specific Data" service parameter of the Stop request is specified. Else it shall be empty.

Table 53 – Structure of Stop_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Stop_ResponsePDU body shall be as specified in Table 54, if the **optional** "Object Specific Data" service parameter of the Stop response is specified. Else it shall be empty.

Table 54 – Structure of Stop_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

4.1.8.1.8 Create

The Create_RequestPDU body shall be as specified in Table 55, if the **optional** "Object Specific Data" service parameter of the Create request is specified. Else it shall be empty.

Table 55 – Structure of Create_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Create_ResponsePDU body shall be as specified in Table 56, if the **optional** "Object Specific Data" service parameter of the Create response is specified. Else it shall be empty.

Table 56 – Structure of Create_ResponsePDU body

Name	Data type
Instance ID	UINT
Object Specific Data	Object/class service- specific

4.1.8.1.9 Delete

The Delete_RequestPDU body shall be as specified in Table 57, if the **optional** "Object Specific Data" service parameter of the Delete request is specified. Else it shall be empty.

Table 57 – Structure of Delete_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Delete_ResponsePDU body shall be as specified in Table 58, if the **optional** "Object Specific Data" service parameter of the Delete response is specified. Else it shall be empty.

Table 58 – Structure of Delete_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

4.1.8.1.10 Get_Attribute_Single

The Get_Attribute_Single_RequestPDU body is empty, except as noted below.

Some CPF 2 profiles support link specific explicit message formats (i.e. they do not support the Message Router request format), which do not enable the Attribute ID to be specified in the Path parameter. For those formats, the Attribute ID is limited to a USINT and is transmitted as Get_Attribute_Single_ResponsePDU body.

The Get_Attribute_ Single_ResponsePDU body shall be as specified in Table 59.

Table 59 – Structure of Get_Attribute_Single_ResponsePDU body

Name	Data type
Attribute Data	Object/class attribute-specific

4.1.8.1.11 Set_Attribute single

The Set_Attribute_Single_RequestPDU body shall be as specified in Table 60.

Table 60 – Structure of Set_Attribute_Single_RequestPDU body

Name	Data type
Attribute Data	Object/class attribute-specific

Some CPF 2 profiles support link specific explicit message formats (i.e. they do not support the Message Router request format), which do not enable the Attribute ID to be specified in the Path parameter. For those formats, the Attribute ID is limited to a USINT and is transmitted as the first parameter (octet) of the Set_Attribute_Single_RequestPDU body.

The Set_Attribute_Single_ResponsePDU body shall be as specified in Table 61, if the **optional** "Object Specific Data" service parameter of the Set_Attribute_Single response is specified. Else it shall be empty.

Table 61 – Structure of Set_Attribute_Single_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

4.1.8.1.12 Find_Next_Object_Instance

The Find_Next_Object_Instance_RequestPDU body shall be as specified in Table 62.

Table 62 – Structure of Find_Next_Object_Instance_RequestPDU body

Name	Data type
Maximum Returned Values	USINT

The Find_Next_Object_Instance_ResponsePDU body shall be as specified in Table 63.

Table 63 – Structure of Find_Next_Object_Instance_ResponsePDU body

Name	Data type
Number Of List Members	USINT
Instance ID List	Array of UINT

4.1.8.1.13 NOP

The NOP_RequestPDU body is empty.

The NOP_ResponsePDU body is empty.

4.1.8.1.14 Apply_Attributes

The Apply_Attributes_RequestPDU body shall be as specified in Table 64, if the **optional** "Object Specific Data" service parameter of the Apply_Attributes request is specified. Else it shall be empty.

Table 64 – Structure of Apply_Attributes_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Apply_Attributes_ResponsePDU body shall be as specified in Table 65, if the **optional** "Object Specific Data" service parameter of the Apply_Attributes response is specified. Else it shall be empty.

Table 65 – Structure of Apply_Attributes_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

4.1.8.1.15 Save

The Save_RequestPDU body shall be as specified in Table 66, if the **optional** "Object Specific Data" service parameter of the Save request is specified. Else it shall be empty.

Table 66 – Structure of Save_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Save_ResponsePDU body shall be as specified in Table 67, if the **optional** "Object Specific Data" service parameter of the Save response is specified. Else it shall be empty.

Table 67 – Structure of Save_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

4.1.8.1.16 Restore

The Restore_RequestPDU body shall be as specified in Table 68, if the **optional** "Object Specific Data" service parameter of the Restore request is specified. Else it shall be empty.

Table 68 – Structure of Restore_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Restore_ResponsePDU body shall be as specified in Table 69, if the **optional** "Object Specific Data" service parameter of the Restore response is specified. Else it shall be empty.

Table 69 – Structure of Restore_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

4.1.8.1.17 Get_Member

The Get_Member_RequestPDU body is empty.

The Get_Member_ResponsePDU body shall be as specified in Table 70.

Table 70 – Structure of Get_Member_ResponsePDU body

Name	Data type
Member ID Member Data	See 4.1.8.1.21 for details

4.1.8.1.18 Set_Member

The Set_Member_RequestPDU body shall be as specified in Table 71.

Table 71 – Structure of Set_Member_RequestPDU body

Name	Data type
Member Data	See 4.1.8.1.21 for details

The Set_Member_ResponsePDU body shall be as specified in Table 72, if the **optional** "Member Data" service parameter of the Set_Member response is specified. Else it shall be empty.

Table 72 – Structure of Set_Member_ResponsePDU body

Name	Data type
Member ID Member Data	See 4.1.8.1.21 for details

4.1.8.1.19 Insert_Member

The Insert_Member_RequestPDU body shall be as specified in Table 73, if the **optional** "Member Data" service parameter of the Insert_Member request is specified. Else it shall be empty.

Table 73 – Structure of Insert_Member_RequestPDU body

Name	Data type
Member Data	See 4.1.8.1.21 for details

The Insert_Member_ResponsePDU body shall be as specified in Table 74. If the **optional** "Member Data" service parameter of the Insert_Member response is not specified, this field shall be empty.

Table 74 – Structure of Insert_Member_ResponsePDU body

Name	Data type
Member ID Member Data	See 4.1.8.1.21 for details

4.1.8.1.20 Remove_Member

The Remove_Member_RequestPDU body is empty.

The Remove_Member_ResponsePDU body shall be as specified in Table 75.

Table 75 – Structure of Remove_Member_ResponsePDU body

Name	Data type
Member ID Member Data	See 4.1.8.1.21 for details

4.1.8.1.21 Common syntax elements for the _Member services

4.1.8.1.21.1 General

Attributes of object classes may consist of arrays of basic data types or structures. To manipulate members of an array, the following member services use the common syntax elements defined in 4.1.8.1.21:

- Get_Member;
- Set_Member;
- Insert_Member;
- Remove_Member.

The first member in an array is specified by a Member ID value of one (1).

4.1.8.1.21.2 Member ID/EX description

Two message formats are defined for member services: basic and extended. Within the extended format, there exist up to 255 protocol options. The message format is selected by the most significant bit of the Member ID.

- When a subnet does not support Attribute and Member level addressing, the Member ID/EX parameter is sent as a 16 bit (WORD) parameter within the service data.
- If the subnet does support Attribute and Member level addressing, the Member ID/EX parameter is sent within the Logical Segment and can be either 8, 16, or 32 bits (SWORD, WORD, DWORD).

Figure 7 defines the bits contained within the Member ID/EX field when sent as a WORD.

Bits							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Member ID Bits 0 to 7							
EX	Member ID Bits 8 to 14						

Figure 7 – Member ID/EX description (WORD)

The lower bits (0 to 14 for a WORD) of the Member ID/EX field identify the member to be manipulated. The highest bit EX (15 for a WORD), selects either the basic (EX=0) or extended (EX=1) message format.

4.1.8.1.21.3 Member request – basic format

Table 76 specifies the common structure basic of the _RequestPDU body for Member services using the basic format.

Table 76 – Common structure of _Member_RequestPDU body (basic format)

Name	Data type	Description of parameter
Attribute ID [a]	USINT	Identifies the attribute the member is to be serviced.
Member ID/ EX [b]	WORD	The lower 15 bits identifies the Member ID of the member to be serviced. If the value is zero (0) the service used determines the behaviour. Extended Protocol Option EX = 0 (bit 15)
Member Data (conditional)	Member specific	This field is required by some services, see individual service descriptions.

[a] This parameter shall be present when the target of the request is on a subnet which does not support explicit message addressing to the attribute level. If the subnet does support attribute level addressing then this parameter shall not be present. In the latter case, the data type of the Attribute ID can be either USINT, UINT, or UDINT.

[b] This parameter shall be present when the target of the request is on a subnet which does not support explicit message addressing to the member or attribute level. If the subnet does support attribute and member level addressing then this parameter shall not be present. In this latter case, the data type of the Member ID/EX can be either SWORD, WORD, or DWORD with the most significant bit being the Extended Protocol Option parameter and the remaining bits being the Member ID.

Table 77 specifies the common structure basic of the _ResponsePDU body for Member services using the basic format.

Table 77 – Common structure of _Member_ResponsePDU body (basic format)

Name	Data type	Description of parameter
Member ID (conditional)	UINT	The Member ID of the member serviced. This field is required by some services, see individual service descriptions. This field is required if the Member Data field is present.
Member Data (conditional)	Member specific	This field is required by some services, see individual service descriptions.

4.1.8.1.21.4 Member request – extended format (general)

If the Extended protocol bit [EX] of the Member ID/EX field is set to a one [1], the octet following the Member ID/EX defines the remainder of the protocol.

Ranges of values for the Extended Protocol ID are divided into three categories (open, vendor specific, reserved), as specified in 4.1.10.1.1.

Table 78 specifies the common structure basic of the _RequestPDU body for Member services using the extended format.

Table 78 – Common structure of _Member_RequestPDU body (extended format)

Name	Data type	Description of parameter
Attribute ID [a]	USINT	Identifies the attribute the member is to be serviced.
Member ID/ EX [b]	WORD	The lower 15 bits identifies the Member ID of the member to be

		serviced. If the value is zero (0) the service used determines the behaviour. Extended Protocol Option EX = 1 (bit 15)
Extended Protocol ID	USINT	Selects the extended protocol used for the remainder of message. See Table 80.
Additional data	Protocol specific	Additional request data is defined by the extended protocol.

[a] This parameter shall be present when the target of the request is on a subnet which does not support explicit message addressing to the attribute level. If the subnet does support attribute level addressing then this parameter shall not be present. In the latter case, the data type of the Attribute ID can be either USINT, UINT, or UDINT.

[b] This parameter shall be present when the target of the request is on a subnet which does not support explicit message addressing to the member or attribute level. If the subnet does support attribute and member level addressing then this parameter shall not be present. In this latter case, the data type of the Member ID/EX can be either SWORD, WORD, or DWORD with the most significant bit being the Extended Protocol Option parameter and the remaining bits being the Member ID.

Table 79 specifies the common structure basic of the _ResponsePDU body for Member services using the extended format.

Table 79 – Common structure of _Member_ResponsePDU body (extended format)

Name	Data type	Description of parameter
Response data	Protocol specific	Response data is defined by the extended protocol option.

Table 80 lists the values currently defined for the Extended Protocol ID.

Table 80 – Extended Protocol ID

Value (hex)	Description
00	Open
01	Multiple Sequential Members
02	International String Selection
03 to 63	Open
64 to C7	Vendor specific
C8 to FF	Reserved
NOTE Reserved ranges may be further defined by ODVA, Inc	

4.1.8.1.21.5 Member request – extended format (Multiple Sequential Members)

When the Extended Protocol ID is set to Multiple Sequential Members [1], the structure of the _RequestPDU body is specified in Table 81.

Table 81 – Structure of _Member_RequestPDU body (Multiple Sequential Members)

Name	Data type	Description of parameter
Attribute ID [a]	USINT	Identifies the attribute the member is to be serviced.
Member ID/ EX [b]	WORD	The lower 15 bits identifies the Member ID of the member to be serviced. If the value is zero (0) the service used determines the behaviour. Extended Protocol Option EX = 1 (bit 15)
Extended Protocol ID	USINT	Selects the Multiple Sequential Members protocol option (value =1).
Number of Members	UINT	Number of members to be serviced.
Member Data (conditional)	Member specific	Multiple sequential Member Data. This field is required by some services, see individual service

Name	Data type	Description of parameter
		descriptions.

a This parameter shall be present when the target of the request is on a subnet which does not support explicit message addressing to the attribute level. If the subnet does support attribute level addressing then this parameter shall not be present. In the latter case, the data type of the Attribute ID can be either USINT, UINT, or UDINT.

b This parameter shall be present when the target of the request is on a subnet which does not support explicit message addressing to the member or attribute level. If the subnet does support attribute and member level addressing then this parameter shall not be present. In this latter case, the data type of the Member ID/EX can be either SWORD, WORD, or DWORD with the most significant bit being the Extended Protocol Option parameter and the remaining bits being the Member ID.

The structure of the _ResponsePDU body is specified in Table 82.

Table 82 – Structure of _Member_ResponsePDU body (Multiple Sequential Members)

Name	Data type	Description of parameter
Number of Members	UINT	Number of members serviced. This field is required
Member ID (conditional)	UINT	Member ID of the first member serviced. This field is required by some services, see individual service descriptions. This field is required if the Member Data field is present.
Member Data (conditional)	Member specific	Multiple sequential Member Data. This field is required by some services, see individual service descriptions.

4.1.8.1.21.6 Member request – extended format (International String Selection)

This service returns a single international character string from an attribute, which has a data type of STRINGI. This service can request a specific language or default to the current active language as specified by the Identity object.

The International String Selection protocol is only valid with the Get_Member service.

The structure of the _RequestPDU body is specified in Table 83.

Table 83 – Structure of _Member_RequestPDU body (International String Selection)

Name	Data type	Description of parameter
Attribute ID	USINT	Identifies the attribute containing the international string.
Member ID/ EX	WORD	The lower 15 bits identifies the language to retrieve. This value is the member ID (array index, starting at one) of the language within the Supported Language List attribute (Attribute ID 12) of the Identity object. Extended Protocol Option EX = 1 (bit 15)
Extended Protocol ID	USINT	Selects the International String Selection protocol option (value = 2).

The structure of the _ResponsePDU body is specified in Table 84.

Table 84 – Structure of _Member_ResponsePDU body (International String Selection)

Name	Data type	Description of parameter
International String	STRUCT of	The character string in the requested language. This parameter follows the definition of a single international character string within the STRINGI data type (see 4.2).
	USINT	The language1 field from the STRINGI data type.

	USINT	The language2 field from the STRINGI data type.
	USINT	The language3 field from the STRINGI data type.
	USINT	The structure of the character string, limited to the values 0xD0, 0xD5, 0xD9 and 0xDA.
	UINT	Character set of the international string.
	OCTET_STRING	International string

4.1.8.1.22 Group_Sync

The Group_Sync_RequestPDU body shall be as specified in Table 85.

Table 85 – Structure of Group_Sync_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Group_Sync_ResponsePDU body shall be as specified in Table 86.

Table 86 – Structure of Group_Sync_ResponsePDU body

Name	Data type	Semantics of values
IsSynchronized	BOOL	Indicates if object is synchronized to the PTP Time Master 0 = Not synchronized 1 = Is synchronized
Object Specific Data	Object/class service- specific	Object/class service- specific

4.1.8.2 Identity object specific syntax elements

4.1.8.2.1 Attributes

4.1.8.2.1.1 Class attributes

The format of the Identity object class attributes shall be as specified in Table 87.

Table 87 – Identity object class attributes

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	Revision of this object. Value = 1
2	Max Instance	UINT	
6	Max ID Number of Class Attributes	UINT	
7	Max ID Number of Instance Attributes	UINT	

4.1.8.2.1.2 Instance attributes

The format of the Identity object instance attributes is as specified in Table 88.

Table 88 – Identity object instance attributes

Attribute ID	Name	Data type	Semantics of values
1	Vendor ID	UINT	The value zero shall not be used.

Attribute ID	Name	Data type	Semantics of values
2	Device Type	UINT	A listing of the Device Type ranges can be found in 4.1.10.2.5
3	Product Code	UINT	The value zero is not valid.
4	Revision	STRUCT of	The value zero shall not be valid for either the Major and Minor Revision fields of a compliant product. A damaged product may return zero to indicate unknown revision if its storage of revision become corrupt.
	Major Revision	USINT	The Major Revision attribute shall be limited to values between 1 and 127 (7 bits). The eighth bit is reserved and shall be zero.
	Minor Revision	USINT	1 to 255
5	Status	WORD	Bit definitions are described in Table 89.
6	Serial Number	UDINT	
7	Product Name	SHORT_STRING	The maximum number of 8-bit characters in this string shall be 32. Each of the characters shall be in the range 0x20 to 0x7E.
8	State	USINT	Present state of the device as represented by the state transition diagram: 0 = Nonexistent 1 = Device Self Testing 2 = Standby 3 = Operational 4 = Major Recoverable Fault 5 = Major Unrecoverable Fault 6 to 254 = Reserved 255 = Default value [a]
9	Configuration Consistency Value	UINT	Contents identify configuration of device
10	Heartbeat Interval	USINT	The nominal interval between heartbeat messages in seconds.
11	Active Language	STRUCT of	Currently active language for the device.
	Language1	USINT	The language1 field from the STRINGI data type.
	Language2	USINT	The language2 field from the STRINGI data type.
	Language3	USINT	The language3 field from the STRINGI data type.
12	Supported Language	ARRAY of STRUCT of	List of languages supported by character strings of data type STRINGI within the device.
	Language1	USINT	The language1 field from the STRINGI data type.
	Language2	USINT	The language2 field from the STRINGI data type.
	Language3	USINT	The language3 field from the STRINGI data type.
13	International Product Name	STRINGI	Names of the product in various languages
14	Semaphore	STRUCT of	Access Synchronization Semaphore
	Vendor Number	UINT	Default: All zero values
	Client Serial Number	UDINT	
	Semaphore Timer	ITIME	Timer valid range 100 thru 32 767
15	Assigned_Name	STRING1	Use assigned name
16	Assigned_Description	STRING1	User assigned description
17	Geographic_Location	STRING1	User assigned location
18	Type15 Identity Info		Reserved for Type 15 devices.

[a] The default value shall be used in the Get_Attribute_All response if the attribute is not implemented.

Table 89 – Identity object bit definitions for status instance attribute

Bit(s)	Called	Definition
0	Owned	1 for TRUE, 0 for FALSE
1		Reserved, set to 0
2	Configured	1 for TRUE, 0 for FALSE
3		Reserved, set to 0
4 to 7	Extended Device Status	Vendor specific, or as specified in Table 90
8	Minor Recoverable Fault	1 for TRUE, 0 for FALSE
9	Minor Unrecoverable Fault	1 for TRUE, 0 for FALSE
10	Major Recoverable Fault	1 for TRUE, 0 for FALSE
11	Major Unrecoverable Fault	1 for TRUE, 0 for FALSE
12 to 15	Extended Device Status 2	Reserved (shall be set to 0) or vendor specific

The values of the status bits 4 to 7 shall be as shown in Table 90.

Table 90 – Default values for extended device status field (bits 4 to 7) of status instance attribute

Value	Meaning
0	Self–Testing (power–up) or State Unknown
1	Firmware Update in progress
2	Communication Fault (at least one lost connection)
3	Unkeyed, Awaiting Connection, no I/O connection established
4	Non–Volatile Configuration Bad
5	Major Fault (see bits 10 and 11 for fault classification and LED states) Minor fault is not a state change
6	Connected, Active, at least one I/O connection in run mode
7	Idle (program mode type), at least one I/O connection established, all in idle mode
8 and 9	reserved, shall not be used
10 to 15	reserved for vendor specific states

4.1.8.2.2 Common services

4.1.8.2.2.1 Get_Attribute_All Response

At the **class level**, the order of the attributes returned in the “Attribute Data” parameter of the Get_Attribute_All response shall be as defined in Table 91.

Table 91 – Class level object/service specific response data of Get_Attribute_All

Attribute ID	Data type	Attribute name	Default value (if not implemented)
1	UINT	Revision	1
2	UINT	Max Instance	1
6	UINT	Max ID Number Class Attributes	0
7	UINT	Max ID Number Instance Attributes	0

Default values shall be inserted for all unsupported attributes.

At the **instance level**, the order of the attributes returned in the “Attribute Data” parameter of the Get_Attribute_All response shall be as defined in Table 92.

Table 92 – Instance level object/service specific response data of Get_Attribute_All

Attribute ID	Data type	Attribute name	Default value (if not implemented)
1	UINT	Vendor ID	
2	UINT	Device Type	
3	UINT	Product Code	
4	STRUCT of:	Revision	
	USINT	Major Revision	
	USINT	Minor Revision	
5	WORD	Status	
6	UDINT	Serial Number	
7	SHORT STRING	Product Name	
8	USINT	State	255
9	UINT	Configuration Consistency Value	0
10	USINT	Heartbeat Interval	0

Default values shall be inserted for all unsupported attributes.

If the Get_Attribute_All service returns data after the Heartbeat Interval and the Identity object Revision is 1, the data after the Heartbeat Interval (octet n+4) shall be ignored (since previous editions of the standard contained unparsable content past that point).

4.1.8.2.2.2 Reset service

The Reset common service shall have the object–specific parameter specified in Table 93.

Table 93 – Object-specific parameter for Reset

Name	Type	Semantics of values
Type	USINT	Type of Reset. See Table 94.

Table 94 – Reset service parameter values

Value	Type of Reset
0	Emulate as closely as possible cycling power on the item the Identity object represents. This value shall be the default if this parameter is omitted.
1	Return as closely as possible to the factory default configuration, then emulate cycling power as closely as possible.
2	Return as closely as possible to the out-of-box configuration with the exception of communication link parameters and emulate cycling power as closely as possible. The communication link parameters that are to be preserved are defined by each network type. See the Reset service of the network specific link object(s) for complete information.
3 to 99	Reserved.
100 to 199	Vendor specific

200 to 255	Reserved

4.1.8.3 Message Router object specific syntax elements

4.1.8.3.1 Attributes

4.1.8.3.1.1 Class attributes

The Message Router object class attributes shall be as specified in Table 95.

Table 95 – Message Router object class attributes

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	Revision of this object. Value = 1
4	Optional attribute list	STRUCT of	
	Number of attributes	UINT	
	Optional attributes	ARRAY of UINT	
5	Optional service list	STRUCT of	
	Number services	UINT	
	Optional services	ARRAY of UINT	
6	Max ID Number of Class Attributes	UINT	
7	Max ID Number of Instance Attributes	UINT	

4.1.8.3.1.2 Instance attributes

The format of the Message Router object instance attributes is as specified in Table 96.

Table 96 – Message Router object instance attributes

Attribute ID	Name	Data type
1	Object_List	STRUCT of
	Number	UINT
	Classes	ARRAY of UINT
2	Number available	UINT

4.1.8.3.2 Common services

Get_Attribute_All Response

At the **class level**, the order of the attributes returned in the “Attribute Data” parameter of the Get_Attribute_All response shall be as specified in Table 97. Default values for all unsupported class attributes shall be as shown in that table.

Table 97 – Class level object/service specific response data of Get_Attribute_All

Class attribute ID	Attribute name, description and default value
1	Revision default = 1
4	Optional attribute list, number of attributes default = 0
5	Optional service list, number of services default = 0
6	Max ID number of class attributes default = 0
7	Max ID number of instance attributes default = 0

Default values for all unsupported instance attributes shall be as shown in Table 98.

Table 98 – Instance level object/service specific response data of Get_Attribute_All

Instance attribute ID	Attribute name, description and default value
1	Object list number, number of supported classes default = 0
2	Number available, maximum number of connections default = 0
3	Always = 0x0000

4.1.8.3.3 Object specific services

Symbolic_Translation

The Symbolic_Translation_RequestPDU body shall be as specified in Table 99.

Table 99 – Structure of Symbolic_Translation_RequestPDU body

Name	Data type
Symbolic Address	Padded EPATH

The Symbolic_Translation_ResponsePDU body shall be as specified in Table 100.

Table 100 – Structure of Symbolic_Translation_ResponsePDU body

Name	Data type
Logical Address	Padded EPATH

The service response may return a status from the list of status codes in Table 101.

Table 101 – Object specific status for Symbolic_Translation service

General Status Code	Extended Status Code	Error Name	Status description
0x20	0x00	Symbolic Path unknown	The Symbolic path is not recognized by the processing node
0x20	0x01	Symbolic Path destination not assigned	The Symbolic path, although recognized, is not presently associated with a Logical path equivalent.
0x20	0x02	Symbolic Path segment error	The symbol identifier or the symbol segment syntax was not understood by the processing node.

4.1.8.4 Assembly object specific syntax elements

4.1.8.4.1 Attributes

4.1.8.4.1.1 Class attributes

The format of the Assembly object class attributes shall be as specified in Table 102.

Table 102 – Assembly object class attributes

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	Revision of this object. Value = 2
2	Max Instance	UINT	

4.1.8.4.1.2 Instance attributes

The format of the Assembly object instance attributes is as specified in Table 103.

Table 103 – Assembly object instance attributes

Attribute ID	Name	Data type	Semantics of values
1	Number of Members in List	UINT	
2	Member List	ARRAY of STRUCT	This attribute has a complex data type
	Member Data Size	UINT	Size in bits
	Member Path Size	UINT	Size in octets (0 = Empty Path)
	Member Path	Packed EPATH	Packed path to the member data. See 4.1.8.9 for definition of Path
3	Data	ARRAY of SWORD	Contain all of the member data packed into one array. This data may contain many different data types. For efficiency it is best to keep this data word aligned by packing it on word boundaries and adding padding as needed. This can be accomplished by using “empty paths” (Member Path Size = 0)
4	Size	UINT	

4.1.8.4.1.3 Assembly Instance ID ranges

The Assembly Instance ID values shall be as shown in Table 104. Assembly Instance ID = 0x00 is reserved and shall not be used.

Table 104 – Assembly Instance ID ranges

Range (hex)	Range (decimal)	Meaning	Quantity
01 to 63	00 to 99	Open[a]	99
64 to C7	100 to 199	Vendor Specific[b]	100
C8 to 2FF	200 to 767	Open[a]	568
300 to 4FF	768 to 1 279	Vendor Specific[b]	512
500 to FFFFF	1 280 to 1 048 575	Open[a]	1 047 296
100000 to FFFFFFFF	1 048 576 to 4 294 967 295	Reserved[c]	4 293 918 720

[a] Static assemblies defined in device profiles.

[b] Static assemblies and dynamic assemblies

[c] Reserved ranges may be further defined by ODVA, Inc.

4.1.8.5 Acknowledge Handler object specific syntax elements

4.1.8.5.1 Attributes

4.1.8.5.1.1 Class attributes

The format of the Acknowledge Handler object class attributes shall be as specified in Table 105.

Table 105 – Acknowledge Handler object class attributes

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	Revision of this object. Value = 1
4	Optional attribute list	STRUCT of	
	Number of attributes	UINT	
	Optional attributes	ARRAY of UINT	
5	Optional service list	STRUCT of	
	Number services	UINT	
	Optional services	ARRAY of UINT	
6	Max ID Number of Class Attributes	UINT	
7	Max ID Number of Instance Attributes	UINT	

4.1.8.5.1.2 Instance attributes

The format of the Acknowledge Handler object instance attributes is as specified in Table 106.

Table 106 – Acknowledge Handler object instance attributes

Attribute ID	Name	Data type	Semantics of values
1	Acknowledge Timer	UINT	Range 1 to 65 535 ms (0 invalid) Default = 16
2	Retry Limit	USINT	Range 0 to 255 Default = 1
3	COS Producing Connection Instance	UINT	Connection Instance ID
4	Ack List Size	SWORD	0 = Dynamic > 0 = Max number of members
5	Ack List	STRUCT of	
		SWORD	Number of members in the array
		ARRAY of UINT	List of Connection Instance IDs
6	Data with Ack Path List Size	SWORD	0 = Dynamic > 0 = Max number of members
7	Data with Ack Path List	STRUCT of	
		SWORD	Number of members in the array
		ARRAY of	
		UINT	Connection Instance ID
		USINT	Path length
		Padded EPATH	Path

4.1.8.5.2 Object specific services

4.1.8.5.2.1 Add_AckData_Path

The Add_AckData_Path_RequestPDU body shall be as specified in Table 107.

Table 107 – Structure of Add_AckData_Path_RequestPDU body

Name	Data type
Connection Instance ID	UINT
Consumer Path Size	USINT
Consumer Path	Padded EPATH

The Add_AckData_Path_ResponsePDU body is empty.

4.1.8.5.2.2 Remove_AckData_Path

The Remove_AckData_Path_RequestPDU body shall be as specified in Table 108.

Table 108 – Structure of Remove_AckData_Path_RequestPDU body

Name	Data type
Connection Instance ID	UINT

The Remove_AckData_Path_ResponsePDU body is empty.

4.1.8.6 Time Sync object specific syntax elements

4.1.8.6.1 Attributes

4.1.8.6.1.1 Class attributes

The format of the Time Sync object class attributes shall be as specified in Table 109.

Table 109 – Time Sync object class attributes

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	Revision of this object. Value = 3

4.1.8.6.1.2 Instance attributes

The format of the Time Sync object instance attributes shall be as specified in Table 110.

Table 110 – Time Sync object instance attributes

Attribute ID	Name	Data type	Semantics of values
1	PTPEnable	BOOL	PTP enabled for this device: 0 = PTP disabled 1 = PTP enabled
2	IsSynchronized	BOOL	Local clock synchronized with master reference clock: 0 = local clock not synchronized 1 = local clock synchronized
3	SystemTimeMicroseconds	ULINT	Current value of system_time in microseconds
4	SystemTimeNanoseconds	ULINT	Current value of system_time in nanoseconds
5	OffsetFromMaster	LINT	Offset between local clock and master clock in nanoseconds
6	MaxOffsetFromMaster	LINT	Maximum offset between local clock and master clock in nanoseconds
7	MeanPathDelayToMaster	LINT	Mean path delay to master in nanoseconds
8	GrandMasterClockInfo	STRUCT of	Grandmaster Clock information
	ClockIdentity	USINT[8]	Unique identifier of the clock. See Table 111 for ClockIdentiy encoding
	ClockClass	UINT	Class of the clock quality. Table 112 shows the values most likely to be used by the Time sync object. See IEC 61588:2009 for a complete list of values
	TimeAccuracy	UINT	Expected absolute accuracy of the clock relative to the PTP epoch. See Table 113 for TimeAccuracy values
	OffsetScaledLogVariance	UINT	Measure of the inherent stability properties of the clock
	CurrentUtcOffset	UINT	Current UTC offset in seconds from International Atomic Time (TAI) of the clock. As of 2006-01-01 at 00:00:00 UTC, the offset was 33 seconds
	TimePropertyFlags	WORD	For the value of the TimePropertyFlags see Table 114
	TimeSource	UINT	For the possible TimeSource values see Table 115
	Priority1	UINT	Relative priority of the grandmaster clock to other clocks in the system. See Priority1 attribute 16
	Priority2	UINT	Relative priority of the grandmaster clock to other clocks in the system. See Priority2 attribute 17
9	ParentClockInfo	STRUCT of	Parent Clock information

Attribute ID	Name	Data type	Semantics of values
	ClockIdentity	USINT[8]	Unique identifier of the clock. See Table 111 for ClockIdentiy encoding
	PortNumber	UINT	Port number of the port identity
	ObservedOffsetScaledLog Variance	UINT	Estimated measure of the parent clock's variance as observed by the slave clock
	ObservedPhaseChangeRate	UDINT	Estimated measure of the parent clock's drift as observed by the slave clock
10	LocalClockInfo	STRUCT of	Local Clock information
	ClockIdentity	USINT[8]	Unique identifier of the clock See Table 111 for ClockIdentiy encoding
	ClockClass	UINT	Class of the clock quality. Table 112 shows the values most likely to be used by the Time sync object. See IEC 61588:2009 for a complete list of values
	Time Accuracy	UINT	Expected absolute accuracy of the clock relative to the PTP epoch. See Table 113 for TimeAccuracy values
	OffsetScaledLogVariance	UINT	Measure of the inherent stability properties of the clock
	CurrentUtcOffset	UINT	Current UTC offset in seconds from International Atomic Time (TAI) of the clock. As of 2006-01-01 at 00:00:00 UTC, the offset was 33 seconds
	TimePropertyFlags	WORD	For the value of the TimePropertyFlags see Table 114
	TimeSource	UINT	For the possible TimeSource values see Table 115
11	NumberOfPorts	UINT	Number of PTP ports on the device
12	PortStateInfo	STRUCT of	Port state information
	NumberOfPorts	UINT	Number of PTP ports on the device
		ARRAY of STRUCT	
	PortNumber	UINT	Number of the port identity
	PortState	UINT	Current state of the PTP port: 1 = Initializing 2 = Faulty 3 = Disabled 4 = Listening 5 = Pre_Master 6 = Master 7 = Passive 8 = Uncalibrated 9 = Slave All other = Reserved
13	PortEnableCfg	STRUCT of	Port enable configuration
	NumberOfPorts	UINT	Number of PTP ports on the device
		ARRAY of STRUCT	
	PortNumber	UINT	Number of the port identity
	PortEnable	UINT	PortEnable has the following values (default is "enabled"): 1 = port enabled 0 = port disabled
14	PortLogAnnounceIntervalCfg	STRUCT of	Port log announce interval configuration
	NumberOfPorts	UINT	Number of PTP ports on the device
		ARRAY of STRUCT	Port log announce interval of each port

Attribute ID	Name	Data type	Semantics of values
	PortNumber	UINT	Number of the port identity
	PortLogAnnounceInterval	UINT	PTP announce interval between successive “Announce” messages issued by a master clock
15	PortLogSyncIntervalCfg	STRUCT of	Port log sync interval configuration
	NumberOfPorts	UINT	Number of PTP ports on the device
		ARRAY of STRUCT	
	PortNumber	UINT	Number of the port identity
	PortLogSyncInterval	UINT	PTP sync interval between successive “Sync” messages issued by a master clock
16	Priority1	USINT	Best Master ranking of this clock and superseeds the clock quality (class, accuracy, and variance). Values are between 0 and 255, wih 0 for the highest priority
17	Priority2	USINT	Best Master ranking of this clock after clock quality (class, accuracy, and variance) has been evaluated and superseeds the tie-breaker. Values are between 0 and 255, wih 0 for the highest priority
18	DomainNumber	USINT	PTP clock domain
19	ClockType	WORD	For ClockType values see Table 116
20	ManufactureIdentity	USINT[4]	Manufacturer identity of the clock. The first 3 octets specify the IEEE OUI (Organization Unique Id) for the manufacturer. The last octet is reserved
21	ProductDescription	STRUCT of	Product description of the device that contains the clock
	Size	UDINT	Size of product description (total number of octets for the Description field) [a]
	Description	USINT[Size]	Product description, formatted as UTF-8 Unicode with a maximum number of symbols of 64.
22	RevisionData	STRUCT of	Revision data of the device that contains the clock
	Size	UDINT	Size of revision data (total number of octets for the Revision field) [a]
	Revision	USINT[Size]	Revision data, formatted as UTF-8 Unicode with a maximum number of symbols of 32.
23	UserDescription	STRUCT of	User description of the device that contains the clock
	Size	UDINT	Size of user description (total number of octets for the Description field) [a]
	Description	USINT[Size]	User description, formatted as UTF-8 Unicode with a maximum number of symbols of 128.
24	PortProfileIdentityInfo	STRUCT of	Port profile identity informationshall be:
	NumberOfPorts	UINT	Number of PTP ports on the device
		ARRAY of STRUCT	
	PortNumber	UINT	Number of the port identity
	PortProfileIdentity	USINT[8]	Port profile identity. The profile identifier is contained in the first 6 octets of the array. The last two octets shall be set to zero. For a CPF 2 profile, profile identifier shall be: 00-21-6C-00-01-00
25	PortPhysicalAddressInfo	STRUCT of	Port physical address information
	NumberOfPorts	UINT	Number of PTP ports on the device

Attribute ID	Name	Data type	Semantics of values
		ARRAY of STRUCT	
	PortNumber	UINT	Number of the port identity
	PhysicalProtocol	USINT[16]	Physical protocol, expressed in ASCII characters. The maximum number of characters is 16. Unused array elements shall be set to zero. See Table 117 for a list of recommended values.
	SizeOfAddress	UINT	Size of the PortPhysicalAddress
	PortPhysicallAddress	USINT[16]	Port physical address, expressed in ASCII characters. The maximum number of characters is 16. Unused array elements shall be set to zero. See Table 117 for a list of recommended values.
26	PortProtocolAddressInfo	STRUCT of	Port protocol address information
	NumberOfPorts	UINT	Number of PTP ports on the device
		ARRAY of STRUCT	
	PortNumber	UINT	Number of the port identity
	NetworkProtocol	UINT	NetworkProtocol values 1 = UDP/IPv4 (e.g. CP 2/2) 2 = UDP/IPv6 3 = ISO/IEC 8802-3 4 = CP 2/3 5 = CP 2/1 0xFFFE = local or unknown protocol all other = reserved
	SizeOfAddress	UINT	Size of the PortProtocolAddress
	PortProtocolAddress	USINT[16]	Port protocol address (e.g. IP address). The maximum number of characters is 16. Unused array elements shall be set to zero.
27	StepRemoved	UINT	Number of communication paths traversed between the local clock and the grandmaster clock
28	SytemTimeAndOffset	STRUCT of	System time and offset
	SystemTime	ULINT	System time in microseconds
	SystemOffset	ULINT	Offset to the local clock value

[a] The number of symbols in the description shall be converted to octets.

Table 111 – ClockIdentity encoding for different network implementations

Network	Octet 0	Octet 1	Octet 2	Octet 3	Octet 4	Octet 5	Octet 6	Octet 7
CP 2/1	0xFF	0x02	Vendor ID [a]		Serial Number [a]			
CP 2/2	MAC Address [b]			0xFF	0xFE	MAC Address [b]		
CP 2/3	0xFF	0x01	Vendor ID [a]		Serial Number [a]			
Local or closed	0xFF	0xFF	Vendor ID [a]		Serial Number [a]			
All others	See IEC 61588:2009							

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

a Most significant octet of Vendor ID and Serial Number is assigned to most significant octet of ClockIdentity. For example, a Vendor ID of "0809" and a Serial Number of "03040506" results in a ClockIdentity for CP 2/3 of "FF 01 08 09 03 04 05 06".

b Most significant octet of MAC Address is assigned to most significant octet of ClockIdentity. For example, a MAC address of "00 01 02 03 04 05" results in a ClockIdentity of "00 01 02 FF FE 03 04 05".

Table 112 – ClockClass values

ClockClass	Value
Primary reference	6
Primary reference (Hold)	7
Degraded reference A (Master only)	52
Degraded reference B (Master / Slave)	187
Default	248
Slave only	255
See IEC 61588:2009	all other

Table 113 – TimeAccuracy values

Clock accuracy	Value
Reserved	0x00 to 0x1F
± 25 ns	0x20
± 100 ns	0x21
± 250 ns	0x22
± 1 µs	0x23
± 2,5 µs	0x24
± 10 µs	0x25
± 25 µs	0x26
± 100 µs	0x27
± 250 µs	0x28
± 1 ms	0x29
± 2,5 ms	0x2A
± 10 ms	0x2B
± 25 ms	0x2C
± 100 ms	0x2D
± 250 ms	0x2E
± 1 s	0x2F
± 10 s	0x30
> ± 10 s	0x31
Reserved	0x32 to 0x7F
For use by alternate PTP profiles	0x80 to 0xFD
Unknown	0xFE
Reserved	0xFF

Table 114 – TimePropertyFlags bit values

TimePropertyFlags	Bit index
Leap indicator 61	0
Leap indicator 59	1
Current UTC offset valid	2
PTP timescale	3
Time traceable	4
Frequency traceable	5

Table 115 – TimeSource values

TimeSource	Value
ATOMIC CLOCK	0x10
GPS	0x20
TERRESTRIAL RADIO	0x30
PTP	0x40
NTP	0x50
HAND SET	0x60
OTHER	0x90
INTERNAL OSCILLATOR	0xA0
For use by alternate PTP profiles	0xF0 to 0xFE
Reserved	0xFF

Table 116 – Types of Clock

Bit Index	Configured clock type
0	Reserved [a]
1	Reserved [a]
2	Reserved [a]
3	Management node
4	End-to-end transparent clock
5	Reserved [a]
6	Boundary clock
7	Ordinary clock
8	Slave only
9 to 15	Reserved [a]

[a] Reserved bits shall be set to 0.

Table 117 – Network protocol to PortPhysicalAddressInfo mapping

Network protocol	PhysicalProtocol	PortPhysicalAddress
UDP/IPv4 (e.g. CP 2/2)	“IEEE 802.3”	Use MAC address
UDP/IPv6	“IEEE 802.3”	Use MAC address
IEEE 802.3	“IEEE 802.3”	Use MAC address
CP 2/1	“ControlNet”	Use MAC ID

Network protocol	PhysicalProtocol	PortPhysicalAddress
CP 2/3	“DeviceNet”	Use MAC ID
Local or unknown protocol	Vendor specific	Use vendor specific

4.1.8.6.2 Common services

Get_Attribute_All Response

At the class level, the Get_Attribute_All response shall contain the class attributes in numerical order, up to the last implemented attribute. Any unimplemented attributes in the response shall use the default attribute values.

4.1.8.7 Parameter object specific syntax elements

4.1.8.7.1 Attributes

4.1.8.7.1.1 Class attributes

The format of the Parameter object class attributes shall be as specified in Table 118.

Table 118 – Parameter object class attributes

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	Revision of this object. Value = 1
2	Max Instance	WORD	The largest instance number of a created object at this class hierarchy level
8	Parameter Class Descriptor	UINT	See Table 119
9	Configuration Assembly Instance	UINT	This attribute shall be set to zero if a configuration assembly is not supported
10	Native Language	UINT	0 = English 1 = French 2 = Spanish 3 = Italian 4 = German 5 = Japanese 6 = Portuguese 7 = Mandarin Chinese

The Parameter Class Descriptor attribute contains bits to describe parameter characteristics. The bits are supported are specified in Table 119.

Table 119 – Parameter Class Descriptor bit values

Bit	Definition
0	Supports Parameter instances 1 = Individual Parameter instances ARE supported. 0 = NO Parameter instances are supported. Only configuration assembly instance used
1	Supports Full attributes 1 = All Full Parameter attributes ARE supported 0 = Only Parameter instance stub attributes are supported
2	Shall do non-volatile storage save command 1 = Shall execute non–volatile storage save command. The Save service of the parameter class is used to save instance parameter values 0 = Need not execute non–volatile storage save command

Bit	Definition
3	Params are stored in Non-Volatile storage 1 = All Full parameters are stored in non–volatile storage 0 = Parameters are not stored in non–volatile storage

4.1.8.7.1.2 Instance attributes

The format of the Parameter object instance attributes is as specified in Table 120.

Table 120 – Parameter object instance attributes

Attribute ID	Name	Data type	Semantics of values
1	Parameter Value	Data type specified in Descriptor, Data Type and Data Size	
2	Link Path Size	USINT	Number of octets
3	Link Path	Packed EPATH	The Link Path is limited to 255 octets
4	Descriptor	WORD	See Table 121
5	Data Type	USINT	See 5.2.2
6	Data Size	USINT	
7	Parameter Name String	SHORT_STRING	The maximum number of characters is 16
8	Units String	SHORT_STRING	The maximum number of characters is 4
9	Help String	SHORT_STRING	The maximum number of characters is 64
10	Minimum Value	Data Type	See Table 122
11	Maximum Value	Data Type	See Table 122
12	Default Value	Data Type	Actual value the parameter should be set to when the user wants the default for the parameter
13	Scaling Multiplier	UINT	See Table 123
14	Scaling Divisor	UINT	See Table 123
15	Scaling Base	UINT	See Table 123
16	Scaling Offset	INT	See Table 123
17	Multiplier Link	UINT	See Table 124
18	Divisor Link	UINT	See Table 124
19	Base Link	UINT	See Table 124
20	Offset Link	UINT	See Table 124
21	Decimal Precision	USINT	Specifies number of decimal places to use when displaying the scaled engineering value. Also used to determine actual increment value so that incrementing a value causes a change in scaled engineering value to this precision
22	International Parameter Name	STRINGI	Human readable string representing the parameter name
23	International Engineering Units	STRINGI	Engineering units associated with the parameter
24	International Help String	STRINGI	Human readable string representing the help string

Table 121 – Semantics of Descriptor Instance attribute

Bit	Definition	Meaning
0	Supports settable path	Indicates that link path can be set
1	Supports Get_Enum_String	Indicates that enumerated strings can be read with the Get_Enum_String service. If descriptor bit 8 is set, enumerated values outside the min/max range may exist
2	Supports scaling	Indicates that the scaling factor should be implemented to present the value to the user in engineering units
3	Supports scaling links	Indicates that the values for the scaling factor may be retrieved from other parameters. If the “Supports Scaling” descriptor bit is also set and the scaling link value is zero, the scaling value shall be used. If the “Supports Scaling” descriptor bit is also set and the scaling link value is not zero, the scaling link value shall be used
4	Read only parameter	Indicates that the Parameter Value attribute can only be read, and not set
5	Monitor parameter	Indicates that the Parameter Value attribute is updated in real time by the device
6	Supports extended precision scaling	Indicates that the extended precision scaling factor shall be implemented to present the value to the user in engineering units
7	Supports non-consecutive enumerated strings	Obsolete
8	Allows both enumeration and min/max range values	Valid for integer data types only. Enumerated values and values in the min/max range are valid. Enumerated values may be inside or outside of the min/max range. If an enumerated value is inside the range, the enumerated string takes precedence over the value
9	Non-displayed parameter	The configuration tool shall not display this parameter
10	Indirect parameter reference	The parameter value represents the instance number of another parameter. The referenced parameter instance shall not be an additional indirect reference. A parameter value of 0 shall indicate no reference. The data type for this parameter shall be USINT, UINT or UDINT
11	Not addressable	The parameter is not directly addressable from the network by any of the Set_Attribute or Get_Attribute services
12	Save supported	This bit, when set, indicates that this parameter may be individually saved by sending the Save service to this parameter instance
13	Apply supported	This bit, when set, indicates the execution of the Apply service to this parameter instance is required before attribute changes affect instance behavior
14	Write only parameter	Indicates that the Parameter Value attribute can only be set, and not read
15	Reserved	This shall be set to zero

Table 122 – Minimum and Maximum Value semantics

Data type	Description and semantics	Minimum Value semantics	Maximum Value semantics	Required/ Optional/Not allowed
OCTET	Bit String – 8 bit length	The least significant bit that is set in the minimum value is the lowest valid bit to be enumerated. The most significant bit that is set in the maximum value is the highest valid bit to be enumerated. A minimum value of zero means that bit 0 is the lowest valid bit to be enumerated. A maximum value of zero is not valid. Example 1 (OCTET type): Min value = 0b00000001 Max value = 0b00001000 Bits 0 to 3 are enumerated Example 2 (WORD TYPE): Min value = 0b0000000000000001		Optional
WORD	Bit String – 16 bit length			
DWORD	Bit String – 32 bit length			
LWORD	Bit String – 64 bit length			

Data type	Description and semantics	Minimum Value semantics	Maximum Value semantics	Required/ Optional/Not allowed
		Max value = 0b0000000000001011 Bits 0 to 3 are enumerated, bits 0 and 1 in max value are ignored, bit 3 in min value is ignored Example 3 (DWORD TYPE): Min value= 0b00000000000000000000000000000000 Max value= 0b00000000000000000000000000001000 Bits 0 to 3 are enumerated, 0 min value implies bit 0		
STRING[a]	String (2 octet length indicator, 1 octet per character)	Minimum string length	Maximum string length	Required
STRING2[a]	String (2 octet length indicator, 2 octets per character)	Minimum string length	Maximum string length	Required
STRINGN[a]	String (2 octet length indicator, N octets per character)	Minimum string length	Maximum string length	Required
SHORT_STRING[a]	Character string (1 octet length indicator, 1 octet characters)	Minimum string length	Maximum string length	Required
STRINGI[a]	International character string	Minimum length (in characters) of the "stringcontents" component of the string's implicit sequence	Maximum length (in characters) of the "stringcontents" component of the string's implicit sequence	Required
EPATH[a]	Encoded Path	Minimum string length	Maximum string length	Optional
ENGUNIT	Engineering Units	The minimum value for this data type is not defined and shall not be specified in the EDS file or the parameter object instance	The maximum value for this data type is not defined and shall not be specified in the EDS file or the parameter object instance	Optional
All other data types		The minimum numeric value that may be assigned to the data value	The maximum numeric value that may be assigned to the data value	Optional[b]

[a] The STRING, STRING2, STRINGN, SHORT_STRING, STRINGI and EPATH data types do not have a minimum or maximum value specification. The minimum and maximum value fields are used to present the minimum and maximum string or path lengths. In these cases, the Data Size parameter is used to represent the number of octets required per character or encoding entry.

[b] If the Minimum Value and/or Maximum Value is not specified then the minimum and/or maximum value for the parameter data value are as defined in IEC 61158-5-2.

Table 123 – Scaling Formula attributes

Attribute	Meaning
Multiplier Value	(Mult) Multiplier value for the scaling formula. This value can be based on another parameter
Divisor Value	(Div) Divisor value for the scaling formula. This value can be based on another parameter specified in the Divisor Link
Base Value	(Base) Base value for the scaling formula. This value can be based on another parameter specified in the Base Link
Offset Value	(Offset) Offset value for the scaling formula. This value can be based on another parameter specified in the Offset Link. This attribute is an INT and can be negative
Decimal Precision	(Precision) Precision value for the scaling formula. This value cannot be based on another parameter

Scaling Links

Scaling values (multiplier, divisor, base, and offset) can also be based on other parameters. Scaling Links specify from which instance that scaling value is to be retrieved. If the scaling link attribute is set to 0, then that scaling value is a constant and not linked to another parameter. Table 124 specifies the scaling links.

Table 124 – Scaling links

Attribute	Meaning
Multiplier Link	Specifies the parameter instance from where the Multiplier Value is retrieved
Divisor Link	Specifies the parameter instance from where the Divisor Value is retrieved
Base Link	Specifies the parameter instance from where the Base Value is retrieved
Offset Link	Specifies the parameter instance from where the Offset Value is retrieved

Scaling Factor Attributes

The Scaling Factor represents actual UINT and INT parameter values in other formats. Formula (1) shall be used to determine the engineering value from the actual value.

$$\text{EngValue} = \frac{(\text{ActualValue} + \text{Offset}) \times \text{Mult} \times \text{Base}}{\text{Div}} \quad (1)$$

The engineering value can then be displayed to the user in the terms specified within the Decimal Precision Instance Attribute. The inverse of the Scaling Formula shall be used to determine the actual value from the engineering value (see Formula (2)).

$$\text{ActualValue} = \frac{(\text{EngValue} \times \text{Div})}{\text{Mult} \times \text{Base}} - \text{Offset} \quad (2)$$

The extended precision scaling factor adds extended precision to the standard scaling factor. Formula (3) shall be used to determine the engineering value from the actual value.

$$\text{EngValue} = \frac{(\text{ActualValue} + \text{Offset}) \times \text{Mult} \times \text{Base}}{\text{Div} \times 10^{\text{Precision}}} \quad (3)$$

The engineering value can then be displayed to the user in the terms specified within the Decimal Precision attribute. The inverse of the extended precision scaling formula shall be used to determine the actual value from the engineering value (see Formula (4)).

$$\text{ActualValue} = \frac{(\text{EngValue} \times \text{Div} \times 10^{\text{Precision}})}{\text{Mult} \times \text{Base}} - \text{Offset} \quad (4)$$

4.1.8.7.2 Common services

Get_Attribute_All Response

At the **class level**, the order of the attributes returned in the “Attribute Data” parameter of the Get_Attribute_All response shall be as defined in Table 125. Default values shall be inserted for all unsupported attributes.

Table 125 – Class level object/service specific response data of Get_Attribute_All

Attribute ID	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Revision (low octet) Default = 1							
	1	Revision (high octet) Default = 0							
2	2	Max Instance (low octet)							
	3	Max Instance (high octet)							
8	4	Parameter Class Descriptor (low octet)							
	5	Parameter Class Descriptor (high octet)							
9	6	Configuration Assembly Instance (low octet)							
	7	Configuration Assembly Instance (high octet)							
10	8	Native Language Default = 0							

At the **instance level**, the setting of the Parameter Class Descriptor attribute dictates what attributes are returned by the Get_Attribute_All service, as follows:

- if this attribute does not have the “Supports Full Attributes” bit set, only the implemented attributes marked Stub (attribute numbers 1 to 6) are returned by the Get_Attribute_All service;
- if this attribute does have the “Supports Full Attributes” bit set,the Get_Attribute_All service returns all implemented attributes (numbers 1 to 24).

For Parameter object stubs, the order of the attributes returned in the “Attribute Data” parameter of the Get_Attribute_All response shall be as defined in Table 126.

Table 126 – Instance level object/service specific response data of Get_Attribute_All (Parameter object stub)

Attribute ID	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Parameter Value (first octet)							
		…							
	n	Parameter Value (last octet)							
2	n+1	Link Path Size							
3	n+1	Link Path (first octet)							
		…							
	…	Link Path (last octet)							
4		Descriptor (low octet)							
		Descriptor (high octet)							
5		Data Type							
6		Data Size							

For full Parameter objects, the order of the attributes returned in the “Attribute Data” parameter of the Get_Attribute_All response shall be as defined in Table 127.

Table 127 – Instance level object/service specific response data of Get_Attribute_All (full Parameter object)

Attribute ID	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Parameter Value (first octet)							
		…							
	n	Parameter Value (last octet)							
2	n+1	Link Path Size							
3	n+1	Link Path (first octet)							
		…							
	…	Link Path (last octet)							
4		Descriptor (low octet)							
		Descriptor (high octet)							
5		Data Type							
6		Data Size							
7		Parameter Name String (charcount) Default = 0							
		Parameter Name String (first octet)							
		…							
		Parameter Name String (last octet)							
8		Units String (charcount) Default = 0							
		Units String (first octet)							
		…							
		Units String (last octet)							
9		Help String (charcount) Default = 0							
		Help String (first octet)							
		…							
		Help String (last octet)							
10		Minimum Value (low octet) Default = 0							
		Minimum Value (high octet) Default = 0							
11		Maximum Value (low octet) Default = 0							
		Maximum Value (high octet) Default = 0							
12		Default Value (low octet) Default = 0							
		Default Value (high octet) Default = 0							
13		Scaling Multiplier (low octet) Default = 1							
		Scaling Multiplier (high octet) Default = 0							
14		Scaling Divisor (low octet) Default = 1							
		Scaling Divisor (high octet) Default = 0							
15		Scaling Base (low octet) Default = 1							
		Scaling Base (high octet) Default = 0							
16		Scaling Offset (low octet) Default = 0							
		Scaling Offset (high octet) Default = 0							
17		Multiplier Link (low octet) Default = 0							
		Multiplier Link (high octet) Default = 0							
18		Divisor Link (low octet) Default = 0							
		Divisor Link (high octet) Default = 0							
19		Base Link (low octet) Default = 0							

Attribute ID	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		Base Link (high octet) Default = 0							
20		Offset Link (low octet) Default = 0							
		Offset Link (high octet) Default = 0							
21		Decimal Precision Default = 0							
22		International Parameter Name[a]							
23		International Engineering Units[a]							
24		International Help String[a]							

[a] For a device that has implemented the International String attributes, the Parameter Name character count, the Units String character count and the Help String character count shall be reported as zero. For devices that do not support the International String attributes, each (and every) International String attribute shall be reported as a single octet of zero, or may be missing from the response. If any International String attribute is supported, all three attributes shall be reported in the response.

4.1.8.7.3 Object specific service

Get_Enum_String

The Get_Enum_String_RequestPDU body shall be as specified in Table 128.

Table 128 – Structure of Get_Enum_String_RequestPDU body

Name	Data type	Semantics of values
Enumerated String Number	USINT	Number of enumerated strings to retrieve. Maximum possible range of values = 0 to 255. Value indicates bit offset (relative to zero) for bit enumerations, actual value (relative to zero) for value enumerations

The Get_Enum_String_ResponsePDU body shall be as specified in Table 129.

Table 129 – Structure of Get_Enum_String_ResponsePDU body

Name	Data type	Semantics of values
Enumerated String	SHORT_STRING	Enumerated strings. Maximum number of characters = 16

Enumerated strings are human–readable strings that describe either a bit or a value, depending on the parameter's data type. The relationship of the string type with the parameter data type is shown in Table 130.

Table 130 – Enumerated strings Type versus Parameter data type

If the parameter's data type is:	Then the enumerated strings returned are:
OCTET, WORD, DWORD, LWORD	Bit enumerated strings
USINT, SINT, UINT, INT, BOOL	Value enumerated strings
All Other data types	Invalid for enumeration

If the parameter's data type is OCTET, WORD, DWORD or LWORD, requesting a Get_Enum_String service with an enumerated string number of 0 on a parameter returns a SHORT_STRING that describes bit 0. Requesting a Get_Enum_String service with an

enumerated string number of 1 on a parameter returns a SHORT_STRING that describes bit 1. This continues up to the maximum bit number supported by the parameter.

If the parameter's data type is SINT, USINT, INT, UINT or BOOL, requesting a Get_Enum_String service with an enumerated string number of 0 on a parameter returns a SHORT_STRING that describes the value of 0. Requesting a Get_Enum_String service with an enumerated string number of 1 returns a SHORT_STRING that describes the value of 1.

4.1.8.8 Connection Manager object specific syntax elements

4.1.8.8.1 Attributes

4.1.8.8.1.1 Class attributes

The format of the Connection Manager object class attributes shall be as specified in Table 131.

Table 131 – Connection Manager object class attributes

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	Revision of this object. Value = 1
2	Max Instance	UDINT	
4	Optional Attribute List	STRUCT of	
	Number attributes	UINT	
	Optional attributes	ARRAY of UINT	

4.1.8.8.1.2 Instance attributes

The format of the Connection Manager object instance attributes is as specified in Table 132.

Table 132 – Connection Manager object instance attributes

Attribute ID	Name	Data type	Semantics of values
1	OpenReqs	UINT	
2	OpenFormat Rejects	UINT	
3	OpenResource Rejects	UINT	
4	OpenOther Rejects	UINT	
5	CloseReqs	UINT	
6	CloseFormat Rejects	UINT	
7	CloseOther Rejects	UINT	
8	ConnTimeouts	UINT	Total number of connection timeouts that have occurred in connections controlled by this Connection Manager.
9	Connection Entry List	STRUCT of	
	NumConnEntries	UINT	Number of bits used in the ConnOpenBits element. This attribute, divided by 8 and incremented for any remainder, gives the length in octets of the array in the ConnOpenBits field.

Attribute ID	Name	Data type	Semantics of values
	ConnOpenBits	ARRAY of BOOL	Bit field. List of connection data. Each bit represents a possible connection. 0 = No Connection. 1 = Connection Established.
10	Reserved / Obsolete		
11	CpuUtilization [a]	UINT	CPU Utilization in tenths of a percent. Range of 0 to 1 000 representing 0 % to 100 %.
12	MaxBuffSize [a]	UDINT	Amount of buffer space is in octets.
13	BufSize Remaining [a]	UDINT	Amount of buffer space is in octets.
[a] The meaning of, and method of calculating, these values are vendor specific.			

4.1.8.8.2 Common services

4.1.8.8.2.1 Get_Attribute_All Response

At the **class level**, the order of the attributes returned in the “Attribute Data” parameter of the Get_Attribute_All response shall be as defined in Table 133.

Table 133 – Class level object/service specific response data of Get_Attribute_All

Attribute ID	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Revision (low octet) Default = 1							
	1	Revision (high octet) Default = 0							
2	2	Max Instance (low octet) Default = 1							
	3	Max Instance (high octet) Default = 0							
6	4	Max ID Number of Class Attributes (low octet) Default = 0							
	5	Max ID Number of Class Attributes (high octet) Default = 0							
7	6	Max ID Number of Instance Attributes (low octet) Default = 0							
	7	Max ID Number of Instance Attributes (high octet) Default = 0							

At the **instance level**, the order of the attributes returned in the “Attribute Data” parameter of the Get_Attribute_All response shall be as defined in Table 134.

Table 134 – Instance level object/service specific response data of Get_Attribute_All

Attribute ID	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	OpenReqs (default = 0)							
	1								
2	2	OpenFormat Rejects (default = 0)							
	3								
3	4	OpenResource Rejects (default = 0)							
	5								
4	6	OpenOther Rejects (default = 0)							
	7								
5	8	CloseReqs (default = 0)							
	9								

Attribute ID	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
6	10	CloseFormat Rejects (default = 0)							
	11								
7	12	CloseOther Rejects (default = 0)							
	13								
8	14	ConnTimeouts (default = 0)							
	15								
9	16	Connection Entry List – NumConnEntries, n octets follow (default = 0)							
	17								
	18	Connection Entry List – ConnOpenBits (up to first 8 BOOL's, if applicable)							
	19	Connection Entry List – ConnOpenBits (up to second 8 BOOL's, if applicable)							
	17+n	Connection Entry List – ConnOpenBits (up to n^{th} 8 BOOL's, if applicable)							
11	18+n	CpuUtilization (default = 0)							
	18+n+1								
12	18+n+2	MaxBuffSize (default = 0)							
	18+n+3								
12	18+n+4								
	18+n+5								
13	18+n+6	BufSize Remaining (default = 0)							
	18+n+7								
12	18+n+8								
	18+n+9								

4.1.8.8.2.2 Set_Attribute_All Request

At the **instance level**, the order of the attributes returned in the “Attribute Data” parameter of the Set_Attribute_All request shall be as defined in Table 135.

Table 135 – Instance level object/service specific request data of Set_Attribute_All

Attribute ID	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	OpenReqs[a]							
	1								
2	2	OpenFormat Rejects[a]							
	3								
3	4	OpenResource Rejects[a]							
	5								
4	6	OpenOther Rejects[a]							
	7								
5	8	CloseReqs[a]							
	9								
6	10	CloseFormat Rejects[a]							
	11								
7	12	CloseOther Rejects[a]							
	13								
8	14	ConnTimeouts[a]							

Attribute ID	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	15								

[a] If this attribute is not supported, the value sent shall be ignored. This condition does not cause the service request to fail.

4.1.8.9 Connection object specific syntax elements

4.1.8.9.1 Attributes

4.1.8.9.1.1 Class attributes

The format of the Connection object class attributes shall be as specified in Table 136.

Table 136 – Connection object class attributes

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	Revision of this object. Value = 1
2	Max Instance	UINT	

4.1.8.9.1.2 Instance attributes

The format of the Connection object instance attributes is as specified in Table 137.

Table 137 – Connection object instance attributes

Attr ID	Name	Data Type	Semantics
1	State	USINT	State of the object.
2	Instance_type	USINT	Indicates either I/O or Messaging Connection.
3	TransportClass_trigger	SWORD	Defines behavior of the Connection.
4	CP2/3_produced_ connection_id	UINT	Placed in ISO 11898 CAN Identifier Field when the Connection transmits on a CP 2/3 subnet.
5	CP2/3_consumed_ connection_id	UINT	ISO 11898CAN Identifier Field value that denotes message to be received on a CP 2/3 subnet.
6	CP2/3_initial_comm_ characteristics	SWORD	Defines the Message Group(s) across which productions and consumptions associated with this Connection occur on a CP 2/3 subnet.
7	Produced_connection_size	UINT	Maximum number of octets transmitted across this Connection.
8	Consumed_connection_size	UINT	Maximum number of octets received across this Connection.
9	Expected_packet_rate	UINT	Defines timing associated with this Connection.
10	CFP2_produced_connection _id	UDINT	Identifies the message sent on the subnet by this connection.
11	CPF2_consumed_connection _id	UDINT	Identifies the message received from the subnet for this connection.
12	Watchdog_timeout_action	USINT	Defines how to handle Inactivity/Watchdog timeouts.
13	Produced_connection_path_ length	UINT	Number of octets in the produced_connection_path attribute.

Attr ID	Name	Data Type	Semantics
14	Produced_connection_path	Packed EPATH	Specifies the application object(s) whose data is to be produced by this Connection object. See 4.1.9.
15	Consumed_connection_path_ length	UINT	Number of octets in the consumed_connection_path attribute.
16	Consumed_connection_path	Packed EPATH	Specifies the application object(s) that are to receive the data consumed by this Connection object. See 4.1.9.
17	Production_inhibit_time	UINT	Defines minimum time between new data production. This attribute is required for all I/O Client connections, except those with a production trigger of Cyclic.
18	Connection_timeout_multiplier	USINT	Specifies the multiplier applied to the expected_packet_rate value to derive the value for the Inactivity/Watchdog Timer.
19	Connection_binding_list	STRUCT UINT Array of UINT	List of I/O connection instances bound to this instance.

State

This attribute defines the current state of the Connection instance. Table 138 defines the possible states and assigns a value used to indicate that state.

Table 138 – Values assigned to the state attribute

Value	State Name	Description
00	Non-existent	The Connection has yet to be instantiated.
01	Configuring	The Connection has been instantiated and is waiting for the following events to occur: (1) to be properly configured and (2) to be told to apply the configuration.
02	Waiting For Connection ID[a]	The Connection instance is waiting exclusively for its consumed_connection_id and/or produced_connection_id attribute to be set.[b]
03	Established	The Connection has been validly/fully configured and the configuration has been successfully applied.
04	Timed Out	If a Connection object experiences an Inactivity/Watchdog timeout, then a transition may be made to this state. See the watchdog_timeout_action attribute description and the description of the Inactivity/Watchdog for more details.
05	Deferred Delete[a]	If an Explicit Messaging Connection object experiences an Inactivity/Watchdog timeout, then a transition may be made to this state. See the watchdog_timeout_action attribute description and the description of the Inactivity/Watchdog Timer for more details.
06	Closing	A Bridged Connection object has received, and is processing, a Forward Close from the Connection Manager. The deletion of the connection does not occur until after a successful Forward Close response has been received from the target node.

[a] This value is only used on CP 2/3.

[b] When the Connection instance attributes are applied it may not be possible for the module to generate the produced_connection_id and/or consumed_connection_id values (see initial_comm_characteristics attribute for rules). If this is the case then all the tasks required to apply the attributes are performed except for the initialization of these attributes within the Connection and associated Link Producer/Consumer objects and a transition is made to this state. In this state the Connection instance is waiting exclusively for its produced_connection_id and/or consumed_connection_id attributes to be set.

Important: A dynamically created connection instance is the child of the Explicit Messaging connection across which it was created. Different subnet types may provide other mechanisms for creating a connection that imply a particular parent-child relationship. See network-specific specifications for details.

Important: All resources associated with a Connection Instance (*A)* that has been dynamically created across an Explicit Messaging Connection (*B*) shall be released if the Explicit Messaging Connection (*B)* times out prior to the dynamically created Connection Instance (*A)* transitioning to the Established state.

Important: When a transition is made to the Established state, all timers associated with the Connection object are activated (see Connection Timing in IEC 61158-5-2, 6.2.3.2.1.7)

Instance_type

This attribute defines the instance type (see Table 139).

Table 139 – Values assigned to the instance_type attribute

Value	Meaning
00	Explicit Messaging. This Connection Instance represents one of the end-points of an Explicit Messaging Connection. An Explicit Messaging Connection is dynamically created by sending the *Open Explicit Messaging Connection Request* to the Connection Class.
01	I/O. This Connection Instance represents one of the end-points of an I/O Connection. An I/O Connection is dynamically created by sending a *Create Request* to the Connection Class.
02	Bridged. This Connection Instance represents an intermediate 'hop' of a bridged I/O or Explicit Messaging connection. A pair of Bridged Connection objects (one for each subnet bridged between) are dynamically created by a node after successfully receiving a Forward Open service to the Connection Manager object when this node is not the end point (target).

TransportClass_trigger

Defines whether this is a producing only, consuming only, or both producing and consuming connection. If this end point is to perform a data production, this attribute also defines the event that triggers the production. The eight (8) bits are divided as shown in Figure 8.

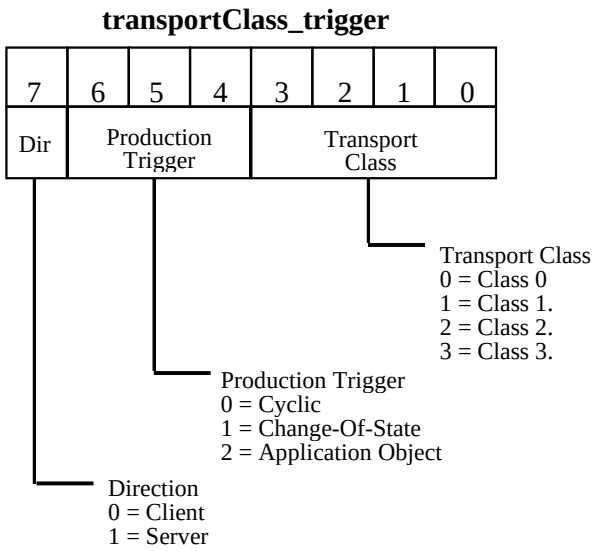


Figure 8 – Transport Class Trigger attribute

The **Direction bit** of the transportClass_trigger octet indicates whether the end-point is to act as the Client or the Server on this connection (see Table 140).

Table 140 – Possible values within Direction Bit

Value	Meaning	
0	Client	This end-point provides the Client behavior associated with this Connection. Additionally, this value indicates that the *Production Trigger* bits within the transportClass_trigger octet contain the description of when the Client is to produce the message associated with this connection. Client connections with production trigger value of 0 or 1 (Cyclic or Change-of-State) shall produce immediately after transitioning to the Established state.
1	Server	This end-point provides the Server behavior associated with this Connection. In addition, this value indicates that the *Production Trigger* bits within the transportClass_trigger octet are to be IGNORED. The *Production Trigger* bits are ignored due to the fact that a Server end-point *reacts* to the transmission from the Client. The only means by which a Server end-point is triggered to transmit is when this *reaction* calls for the production of a message (Transport Classes 2 or 3).

Table 141 lists the values that are possible within the **Production Trigger** bits of the **transportClass_trigger** attribute.

Table 141 – Possible values within Production Trigger Bits

If the value is:	Then the Production of a message is:	
0	Cyclic	The expiration of the Transmission Trigger Timer triggers the data production. See IEC 61158-5-2, 6.2.3.2.1.7 Connection Timing for a detailed description of the Transmission Trigger Timer.
1	Change-Of-State	Production occurs when a change-of-state is detected by the application object. NOTE The consuming end-point may have been configured to expect the packet at a certain rate, regardless of the triggering mechanism at the producing end-point. See the description of the expected_packet_rate attribute of a Connection object and the description of Connection Timing in IEC 61158-5-2, 6.2.3.2.1.7 for more information.
2	Application Object Triggered	The application object decides when to trigger the production. NOTE The consuming end-point may have been configured to expect the packet at a certain rate, regardless of the triggering mechanism at the producing end-point. See the description of the expected_packet_rate attribute of a Connection object and the description of Connection Timing in IEC 61158-5-2, 6.2.3.2.1.7 for more information.
3 - 7	Reserved	

Table 142 lists possible values within the **Transport Class** nibble of the **transportClass_trigger** attribute. Behaviors resulting from these particular values are illustrated in the series of figures that follow the table.

Table 142 – Possible values within Transport Class Bits

Value	Meaning	
0	Transport Class 0	Based on the value within the *Dir* bit, this connection end-point will be a producing only OR consuming only end-point. Upon application of this Connection instance, the module instantiates either a Link Producer (*Dir* bit = Client, producing only) or a Link Consumer (*Dir* bit = Server, consuming only) to be associated with this Connection.
1	Transport Class 1	
2	Transport Class 2	Indicates that the module will both produce AND consume across this connection. The Client end-point generates the first data production that is consumed by the Server, which causes the Server to return a production that is consumed by the Client.
3	Transport Class 3	
4	Transport Class 4	Non-blocking.
5	Transport Class 5	Non-blocking, fragmenting.
6	Transport Class 6	Multicast, fragmenting.
7 - F	Reserved	

A 16-bit sequence count value is prepended to all Class 1, 2, and 3 transports. This value is used to detect delivery of duplicate data packets. Sequence count values are initialized on the first message production and determined by the transport and trigger type on subsequent message productions. A resend of old data to maintain the connection shall not cause the sequence count to change and a consumer shall ignore data when it is received with a duplicate sequence count. Consuming applications can use this mechanism to distinguish between new samples and old samples that were sent to maintain the connection.

However, on CP 2/3, transport classes 2 and 3 do not prepend a 16-bit sequence count as described in 4.1.4.

The following tables and figures illustrate the valid combinations of **Production Trigger** and **Transport Class** and provides a description of the Client and Server behaviors. See IEC 61158-5-2, 6.2.3.2.1.7 for a description of the **Transmission Trigger Timer,** which is shown in the illustrations.

Table 143 summarizes the valid values for the **transportClass_trigger** attribute of a Connection Instance:

Table 143 – TransportClass_Trigger attribute values summary

TransportClass_trigger bits	Meaning
1 xxx 0000	Direction = Server, Production Trigger = IGNORED, Transport Class = 0.
1 xxx 0001	Direction = Server, Production Trigger = IGNORED, Transport Class = 1.
1 xxx 0010	Direction = Server, Production Trigger = IGNORED, Transport Class = 2.
1 xxx 0011	Direction = Server, Production Trigger = IGNORED, Transport Class = 3. This is the value assigned to this attribute within the Server end-point of an Explicit Messaging Connection.
0 000 0000	Direction = Client, Production Trigger = Cyclic, Transport Class = 0.
0 000 0001	Direction = Client, Production Trigger = Cyclic, Transport Class = 1.
0 000 0010	Direction = Client, Production Trigger = Cyclic, Transport Class = 2.
0 000 0011	Direction = Client, Production Trigger = Cyclic, Transport Class = 3.
0 001 0000	Direction = Client, Production Trigger = Change-Of-State, Transport Class = 0.
0 001 0001	Direction = Client, Production Trigger = Change-Of-State, Transport Class = 1.

TransportClass_trigger bits	Meaning
0 001 0010	Direction = Client, Production Trigger = Change-Of-State, Transport Class = 2.
0 001 0011	Direction = Client, Production Trigger = Change-Of-State, Transport Class = 3.
0 010 0000	Direction = Client, Production Trigger = Application object, Transport Class = 0.
0 010 00001	Direction = Client, Production Trigger = Application object, Transport Class = 1
0 010 0010	Direction = Client, Production Trigger = Application object, Transport Class = 2.
0 010 0011	Direction = Client, Production Trigger = Application object, Transport Class = 3. This is the value assigned to this attribute within the Client end-point of an Explicit Messaging Connection.
1 111 1111	Default value assigned to this attribute within an I/O Connection.

Table 144 summarizes client behavior for Class 0 in regards to message production for the different trigger types.

Table 144 – Transport Class 0 client behavior summary

Trigger type	Production triggered by	Repeated production to maintain connection
Cyclic	Transmission Timer (API)	n/a
Change of State	Detected data change	Transmission Timer (API)
Application	Application update of data	Transmission Timer (API)

Table 145 summarizes client behavior for Classes 1, 2 and 3 in regards to message production and sequence count for the different trigger types.

Table 145 – Transport Class 1, 2 and 3 client behavior summary

Trigger type	Sequence count incremented by	Production triggered by	Repeated production to maintain connection
Cyclic	Application update of data	Transmission Timer (API)	n/a
Change of State	Detected data change	Detected data change	Transmission Timer (API)
Application	Application update of data	Application update of data	Transmission Timer (API)

CP2/3_produced_connection_id

This attribute is Required for a CP 2/3 subnet. Other subnet types shall not use this attribute.

It contains the CP 2/3 Connection ID to be associated with transmissions sent across this connection (if any), i.e. the value that will be specified in the CAN Identifier Field of ISO 11898 when this Connection transmits. See IEC 62026-3:2008, 5.1.2 for a description of how CP 2/3 uses the CAN Identifier Field. This value is loaded directly into the associated Link Producer's connection_id attribute. Values are defined in Table 146.

Table 146 – Values defined for the CP2/3_produced_connection_id attribute

Value	Meaning
0 - $7F0_{hex}$	The value to be placed in the CAN Identifier Field when this Connection transmits.
800_{hex} - $FFFE_{hex}$	Reserved

Value	Meaning
$FFFF_{hex}$	Default value assigned to this attribute within an I/O Connection. This attribute will retain this value if this Connection instance is not producing any data (consumer only).

CP2/3_consumed_connection_id

This attribute is Required for a CP 2/3 subnet. Other subnet types shall not use this attribute.

It contains the Connection ID which identifies messages to be received across this connection (if any), i.e. the CAN Identifier Field value that is associated with messages this Connection object receives. See IEC 62026-3:2008, 5.1.2 for a description of how CP 2/3 uses the CAN Identifier Field. This value is loaded directly into the associated Link Consumer's connection_id attribute. Values are defined in Table 147.

Table 147 – Values defined for the CP2/3_consumed_connection_id attribute

Value	Meaning
0 - $7F0_{hex}$	The value that identifies messages to be consumed. This will be specified in the CAN Identifier Field of messages that are to be consumed.
800_{hex} - $FFFE_{hex}$	Reserved
$FFFF_{hex}$	Default value assigned to this attribute within an I/O Connection. This attribute will retain this value if this Connection instance is not consuming any data (producer only).

CP2/3_initial_comm_characteristics

This attribute is Required for a CP 2/3 subnet. Other subnet types shall not use this attribute.

It defines the Message Group(s) across which productions and consumptions associated with this Connection occur.

This octet is divided into two nibbles, as show in Figure 9.

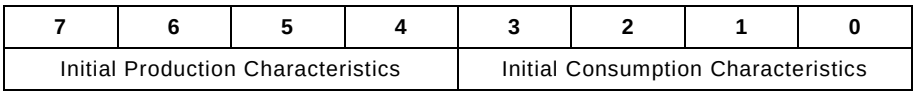


Figure 9 – CP2/3_initial_comm_characteristics attribute format

Table 148 lists the values that are possible within the Initial Production Characteristics nibble (upper nibble) of the CP2/3_initial_comm_characteristics attribute.

Table 148 – Values for the Initial Production Characteristics nibble

Value	Meaning	
0	Produce across Message Group 1	The production associated with this Connection is to take place across Message Group 1. The producing module generates the Connection ID value and loads it into the Connection object's CP2/3_produced_connection_id attribute. The producing module allocates a Message ID from its Group 1 Message ID pool and combines this with its Source MAC ID to generate the Connection ID. The numerically lowest available Group 1 Message ID is to be used in generating the CP2/3_produced_connection_id attribute value. This value shall also be loaded into the corresponding CP2/3_consumed_connection_id attribute(s) associated with the consuming Connection object(s).
1	Produce across Message Group 2 (Destination)	The production associated with this Connection is to take place across Message Group 2. Additionally, the intended recipient's MAC ID (Destination MAC ID) is to be placed within the MAC ID component of the Group 2 Identifier Field. In this case, the consuming module generates the Connection ID value to be associated with transmissions across this connection. When the consuming module has generated this value and loaded it into the appropriate Connection object's CP2/3_consumed_connection_id attribute, it can be read and subsequently loaded into the producing Connection object's CP2/3_produced_connection_id attribute.
2	Produce across Message Group 2 (Source)	The production associated with this Connection is to take place across Message Group 2. In addition, the producing module's MAC ID (Source MAC ID) is to be placed within the MAC ID component of the Group 2 Identifier. In this case, the producing module generates the Connection ID value and loads it into the Connection object's CP2/3_produced_connection_id attribute. The numerically lowest available Group 2 Message ID is to be used in generating the CP2/3_produced_connection_id attribute value. This value shall also be loaded into the corresponding CP2/3_consumed_connection_id attribute(s) associated with the consuming Connection object(s).
3	Produce across Message Group 3	The production associated with this Connection is to take place across Message Group 3. The producing module generates the Connection ID value and loads it into the Connection object's CP2/3_produced_connection_id attribute. The producing module allocates a Message ID from its Group 3 Message ID pool and combines this with its Source MAC ID to generate the Connection ID. The numerically lowest available Group 3 Message ID is to be used in generating the CP2/3_produced_connection_id attribute value. This value shall also be loaded into the corresponding CP2/3_consumed_connection_id attribute(s) associated with the consuming Connection object(s).
4 - 0xE	Reserved	
0xF	Default value	The default value assigned to the CP2/3 Initial Production Characteristics nibble within an I/O Connection. NOTE If this is a consuming only I/O Connection, then the default value remains in this nibble. Explicit Messaging Connection objects automatically configure this attribute when the Connection is established.

Table 149 lists the possible values within the Initial Consumption Characteristics nibble (lower nibble) of the CP2/3_initial_comm_characteristics attribute.

Table 149 – Values for the Initial Consumption Characteristics nibble

Value	Meaning	
0	Consume a Group 1 Message	The message to be consumed will be transmitted across Message Group 1. The producing module generates the Connection ID value. This value shall be loaded into the CP2/3_consumed_connection_id attribute associated with the consuming Connection object(s).
1	Consume a Group 2 Message (Destination)	The message to be consumed will be transmitted across Message Group 2. The intended recipient's MAC ID (Destination MAC ID) is specified within the Group 2 Identifier. The consuming module generates the Connection ID value and loads it into the CP2/3_consumed_connection_id attribute associated with this Connection object. The numerically lowest available Group 2 Message ID is to be used in generating the CP2/3_consumed_connection_id attribute value. This value shall be loaded into the producing Connection object's CP2/3_produced_connection_id attribute.
2	Consume a Group 2 Message (Source)	The message to be consumed will be transmitted across Message Group 2. The transmitting module's MAC ID (Source MAC ID) is specified within the Group 2 Identifier. In this case, the producing module generates the Connection ID value and loads it into the Connection object's the CP2/3_produced_connection_id attribute. This value shall be loaded into the CP2/3_consumed_connection_id attribute associated with the consuming Connection object(s).
3	Consume a Group 3 Message	The message to be consumed will be transmitted as a Group 3 Message. The producing module generates the Connection ID value. The Connection ID value shall be loaded into this Connection object's CP2/3_consumed_connection_id attribute.
4 - 0xE	Reserved	
0xF	Default value	The default value assigned to the CP 2/3 Initial Consumption Characteristics nibble within an I/O Connection. NOTE If this is a producing only I/O Connection, then the default value remains in this nibble. Explicit Messaging Connections automatically configure this attribute when the Connection is established.

Important: The module that generates a Connection ID shall guarantee that it does not allocate the Message ID/MAC ID pair in such a way that two separate modules are capable of transmitting identical bit patterns within the Identifier Field.

Produced_connection_size

The meaning of this attribute is different for Explicit Messaging Connections than it is for I/O Connections. If the subnet defines a fragmentation protocol and the device supports fragmentation, this size may be larger than the largest frame size. See the network specific adaptation specifications for more details.

For Explicit Messaging Connections:

This attribute signifies the maximum number of Message Router Request/Response Data octets (see 4.1.8) that a device is able to transmit across this Connection. Devices that place a known limit on the maximum amount of Message Router Request/Response Data that can be transmitted in a single message, or single fragmented series initialize this attribute accordingly. Devices that cannot or do not predefine an up-front transmit limit place the value 0xffff into this attribute (there may still be a limit, however, it is not known in advance).

Important: Due to the nature of Explicit Messaging, the length of Explicit Messages will fluctuate over the lifetime of a connection. Explicit Messaging Connections perform fragmentation based on the length of the current message to transmit where the subnet type may define a fragmentation protocol.

For I/O Connections:

If the **transportClass_trigger** indicates that this Connection instance is to produce, then this attribute defines the maximum amount of I/O data that may be produced as a single unit across this connection. The amount of I/O to be transmitted at any given point in time can be less than or equal to the **connection_size** attribute.

This attribute defaults to zero (0) within an I/O Connection. If the subnet type supports fragmentation and this attribute is set to a value greater than the largest payload in an I/O Connection, then the Connection will break up the data into multiple fragments. See the network specific adaptation specifications for more details.

Important: Fragmentation within I/O Connections is performed based on the value within this attribute, regardless of the current amount of data to transmit.

Important: I/O Messages that contain no Application I/O Data and were configured to contain data (via produced_connection_size being greater than zero (0)) are defined to indicate a **No Data** event for the receiving application object(s). The behavior of an application object upon detection of the **No Data** event is application object specific.

Consumed_connection_size

The meaning of this attribute is different for Explicit Messaging Connections than it is for I/O Connections. If the subnet defines a fragmentation protocol and the device supports fragmentation, this size may be larger than the largest frame size. See the network specific adaptation specifications for more details.

For Explicit Messaging Connections:

This attribute signifies the maximum number of Message Router Request/Response Data octets that a device is able to receive across this Connection.

Devices that place a known limit on the maximum amount of Message Router Request/Response Data that can be received in a single message, or single fragmented series initialize this attribute accordingly. Devices that cannot or do not predefine an up-front receive limit place the value 0xffff into this attribute (there may still be a limit, however, it is not known in advance). Because of the nature of Explicit Messaging, the length of Explicit Messages will fluctuate over the lifetime of a connection.

For I/O Connections:

If the **transportClass_trigger** attribute indicates that this Connection is to consume, then this attribute defines the maximum amount of data that may be received as a single unit across this connection. The actual amount of I/O data received at any given time can be less than or equal to the **connection_size** attribute.

This attribute defaults to zero (0) within an I/O Connection. If the subnet type supports fragmentation and this attribute is set to a value greater than the largest payload in an I/O Connection**,** then the Connection will process the fragmentation protocol. See the network specific adaptation specifications for more details.

The length of an I/O Message shall be less than or equal to this attribute for an I/O Connection object to receive it as a valid message. If an I/O Connection object receives a message whose length is greater than this attribute, then it immediately discards the message and discontinues any subsequent processing.

NOTE With respect to the Server end-point of a Transport Class 2 or 3 Connection, this error condition (too much data) results in no response being transmitted and the watchdog timer is not kicked.

Expected_packet_rate

This attribute is used to generate the values loaded into the **Transmission Trigger Timer** and the **Inactivity/Watchdog Timer**. See IEC 61158-5-2, 6.2.3.2.1.7 for a description of the Transmission Trigger and Inactivity/Watchdog timers.

The resolution of this attribute is in milliseconds. A request to configure this attribute may result in the specification of a time value that a product cannot meet. In addition to performing product specific range checking when a request to modify this attribute is received, the following steps are performed:

- If the specified value is not equal to an increment of the available clock resolution, then the value is rounded up to the next serviceable value. For example: a Set_Attribute_Single request is received specifying the value 5 for the **expected_packet_rate** attribute and the product provides a 10 millisecond resolution on timers. In this case the product would load the value 10 into the **expected_packet_rate** attribute.
- The value that is actually loaded into the **expected_packet_rate** attribute is reported in the Service Data Field of a Set_Attribute_Single response message associated with a request to modify this attribute.
- If the requested value is equal to an increment of the clock resolution, then the requested value is loaded into the expected_packet_rate and reported in the response. For example: if the value 100 is requested and the clock resolution is 10 milliseconds, then of a value of 100 is loaded.
- When a Connection object is in the **Established** state, any modifications to the **expected_packet_rate** attribute have immediate effect on the Inactivity/Watchdog Timer. The following steps are performed by a Connection object in the **Established** state when a request is received to modify the **expected_packet_rate** attribute:
  - the current Inactivity/Watchdog Timer is canceled,
  - a new Inactivity/Watchdog Timer is activated based on the new value in the expected_packet_rate attribute.

This attribute defaults to 2500 (2 500 ms) within Explicit Messaging Connections, and to zero (0) within an I/O Connection.

CPF2_produced_connection_id

Contains the Connection ID, which identifies messages to be sent across this connection (if any). This attribute shall not be implemented when the subnet type is CP 2/3.

CPF2_consumed_connection_id

Contains the Connection ID, which identifies messages to be received across this connection (if any). This attribute shall not be implemented when the subnet type is CP 2/3.

Watchdog_timeout_action

This attribute defines the action the Connection object should perform when the Inactivity/Watchdog Timer expires. Table 150 defines the specifics of this attribute.

Table 150 – Values for the watchdog_timeout_action

Value	Meaning
0	*Transition to Timed Out*. The Connection transitions to the Timed Out state and remains in this state until it is Reset or Deleted. The command to Reset or Delete could come from an internal source (e.g., an application object) or could come from the network (e.g., a configuration tool). This is the default value for this attribute with respect to I/O Connections. This value is invalid for Explicit Messaging Connections.
1	*Auto Delete*. The Connection Class automatically deletes the Connection if it experiences an Inactivity/Watchdog timeout. This is the default value for this attribute with respect to Explicit Messaging Connections.
2	*Auto Reset*. The Connection remains in the Established state and immediately restarts the Inactivity/Watchdog timer. This value is invalid for Explicit Messaging Connections.
3	*Deferred Delete*. The Connection transitions to the Deferred state if any child connection instances are in the Established state. If no child connection instances are in the Established state the connection is deleted. This value is only used on CP 2/3 and is invalid for I/O Messaging Connections.
4 - FF	Reserved

Produced_connection_path_length

Specifies the number of octets of information within the **produced_connection_path** attribute. This is automatically initialized when the **produced_connection_path** attribute is configured. This attribute defaults to the value zero (0).

Produced_connection_path

The **produced_connection_path** attribute is made up of a octet stream which defines the application object(s) whose data is to be produced by this Connection object. The format of this octet stream is specified in 4.1.9. This attribute defaults to being empty upon instantiation of the Connection. It remains empty within Explicit Messaging Connections and within Connection objects that do not produce.

Consumed_connection_path_length

Specifies the number of octets of information within the **consumed_connection_path** attribute. This is automatically initialized when the **consumed_connection_path** attribute is configured. This attribute defaults to the value zero (0).

Consumed_connection_path

The **consumed_connection_path** attribute is made up of an octet stream which defines the application object(s) that are to receive the data consumed by this Connection object. The format of this octet stream is specified in 4.1.9. This attribute defaults to being empty upon instantiation of the Connection. It remains empty within Explicit Messaging Connections and within Connection objects that do not consume.

Production_inhibit_time

This attribute is used to configure the minimum delay time between new data production. This is required for all I/O Client connections, except those with a production trigger of Cyclic. The Set_Attribute_Single service shall be supported when this attribute is implemented. A value of zero (the default value for this attribute) indicates no inhibit time.

The resolution of this attribute is in milliseconds. A request to configure this attribute may result in the specification of time value that a product cannot meet. In addition to performing product specific range checking when a request to modify this attribute is received, the following steps are performed:

- If the specified value is not equal to an increment of the available clock resolution, then the value is rounded up to the next serviceable value. For example: a Set_Attribute_Single request is received specifying the value 5 for the **production_inhibit_time** attribute and the product provides a 10 millisecond resolution on times. In this case the product would load the value 10 into the **production_inhibit_time** attribute.
- The value that is actually loaded into the **production_inhibit_time** attribute is reported in the Service Data Field of a Set_Attribute_Single response message associated with a request to modify this attribute.
- If the requested value is equal to an increment of the clock resolution, then the requested value is loaded into the **production_inhibit_time** and reported in the response. For example: the value 100 is requested and the clock resolution is 10 milliseconds.

The **production_inhibit_time** value is loaded in the Production Inhibit Timer each time new data production occurs.

When a Connection object is in the **Established** state, any modifications to the **production_inhibit_time** attribute have no effect on a currently running Production Inhibit Timer. The new production_inhibit_time value is loaded into the Production Inhibit Timer on the following new data production.

When the apply_attributes service is received, the **production_inhibit_time** shall be verified against the **expected_packet_rate** attribute. If the **expected_packet_rate v**alue is greater than zero, but less than the **production_inhibit_time** value, then an error shall be returned. In this case, where two attribute values conflict use the **production_inhibit_time attribute** ID as the additional error code returned in the error response.

Connection_timeout_multiplier

The Connection_timeout_multiplier specifies the multiplier applied to the expected packet rate to obtain the value used by the Inactivity/Watchdog Timer. See the Connection Timeout Multiplier parameter description within the Connection Manager object Specific Service Parameters for the enumerated values of this attribute. The default value for this attribute is zero (specifying a multiplier of 4).

Connection_binding_list

The Connection_binding_list attribute identifies connection instances that are bound to this connection. The attribute structure provides a 16 bit count of bound connections, followed by a list of the bound instances. This attribute shall be supported if the Connection_Bind service is supported.

4.1.8.9.2 Object specific services

4.1.8.9.2.1 Connection_Bind Service

The Connection_Bind object specific service binds two dynamically created I/O connection instances together for purposes of connection timeouts and deletions. A dynamically created I/O connection is the result of a Create service to the Connection class with the Instance_type attribute set to I/O (attribute value of 1). If one of these I/O connection instance times out or is deleted, the other instance shall exhibit the same behavior.

The shall be as specified in Table 151.

Table 151 – Structure of Connection_Bind_RequestPDU body

Parameter Name	Data Type	Parameter Description
Bound Instances	STRUCT of UINT UINT	Instance numbers of Connection objects to be bound.

The Connection_Bind_ResponsePDU body is empty.

The service response may return a status from the list of status codes in Table 152.

Table 152 – Object specific status for Connection_Bind service

General Status Code	Extended Status Code	Status Description
0x02	0x01	One or both of the connection instances is non-existent.
0x02	0x02	The connection class and/or instance is out of resources to bind instances.
0x0C	0x01	Both of the connection instances are existent, but at least one is not in the Established state.
0x20	0x01	Both connection instances are the same value.
0xD0	0x01	One or both of the connection instances is not a dynamically created I/O connection.
0xD0	0x02	One or both of the connection instances were created internally and the device is not allowing a binding to it.

4.1.8.9.2.2 Producing_Application_Lookup Service

The Producing_Application_Lookup object specific service provides a mechanism to find one or more connection instances in the Established state producing data from a given application object. If the requested producing application path is being produced by any connection in the Established state, the instance number of each connection producing from that path is returned.

The Producing_Application_Lookup_RequestPDU body shall be as specified in Table 153.

Table 153 – Structure of Producing_Application_Lookup_RequestPDU body

Parameter Name	Data Type	Parameter Description
Producing Application Path	Packed EPATH	Connection path of producing application to be searched for within connections in the Established state. The path shall be a single Logical or Symbolic path.

The Producing_Application_Lookup_ResponsePDU body shall be as specified in Table 154.

Table 154 – Structure of Producing_Application_Lookup_ResponsePDU body

Parameter Name	Data Type	Parameter Description
Instance Count	UINT	Number of Instances returned in the Connection Instance List parameter.
Connection Instance List	Struct of UINT	List of instance numbers of connection producing the data from the requested connection path within the node.

The service response may return a status from the list of status codes in Table 155.

Table 155 – Producing_Application_Lookup Service status codes

General Status Code	Extended Status Code	Status Description
0x02	0x01	The connection path was not found in any connection instance in the Established state.

4.1.8.9.2.3 SafetyOpen Service

See IEC 61784-3-2 for the definition of this service.

4.1.8.9.2.4 SafetyClose Service

See IEC 61784-3-2 for the definition of this service.

4.1.9 Message and connection paths

4.1.9.1 Contents

The path shall specify the object element that is either the target of a connection request, or the destination of a message request. The path shall contain multiple segments which can indicate the route to the next node (in the case of multiple links), what to connect to or where to send a message in the target device. Each segment shall be comprised of a segment descriptor octet which specifies the segment type and format, and segment information whose size is dependent on the segment type/format.

A path has a data type EPATH (Packed or Padded). The two types of path shall be

- padded paths (indicated as data type Padded EPATH);
- packed paths (indicated as data type Packed EPATH).

These two path formats (padded and packed) shall not be interchangeable. Usage of padded versus packed shall be specified, depending on the context of use.

Each segment of a padded path shall be 16 bit word aligned. If a pad octet is required to achieve the alignment, then the segment shall specify the location of the pad octet. A packed path shall not contain any pad octets.

The possible segment types shall be as follows:

Port segment (where);

Logical segment (what);

Network segment (when or how);

Symbolic segment;

Data segment.

4.1.9.2 Segment type

Each segment of the path shall contain a segment type/format octet to indicate how the segment is to be interpreted. The segment type/format is contained in the first octet of the segment. The bits of the first octet of a path segment shall be numbered 0 to 7, where bit 0 shall be the least significant bit of the octet. Bits of the segment type are defined in Figure 10.

Segment Type			Segment Format					
7	6	5	4	3	2	1	0	
0	0	0	Port Segment					
0	0	1	Logical Segment					
0	1	0	Network Segment					
0	1	1	Symbolic Segment					
1	0	0	Data Segment					
1	0	1	Data Type (constructed, see 5.2.3)					
1	1	0	Data Type (elementary, see 5.2.2)					
1	1	1	Reserved for future use					

Figure 10 – Segment type

The meaning of the Segment Format bits is based on the specified Segment Type.

4.1.9.3 Port segment

The port segment shall indicate

- communication port through which to leave the node (expressed as a 16-bit number);
- link address of the next device in the path.

Figure 11 shows the overall structure of the port segment.

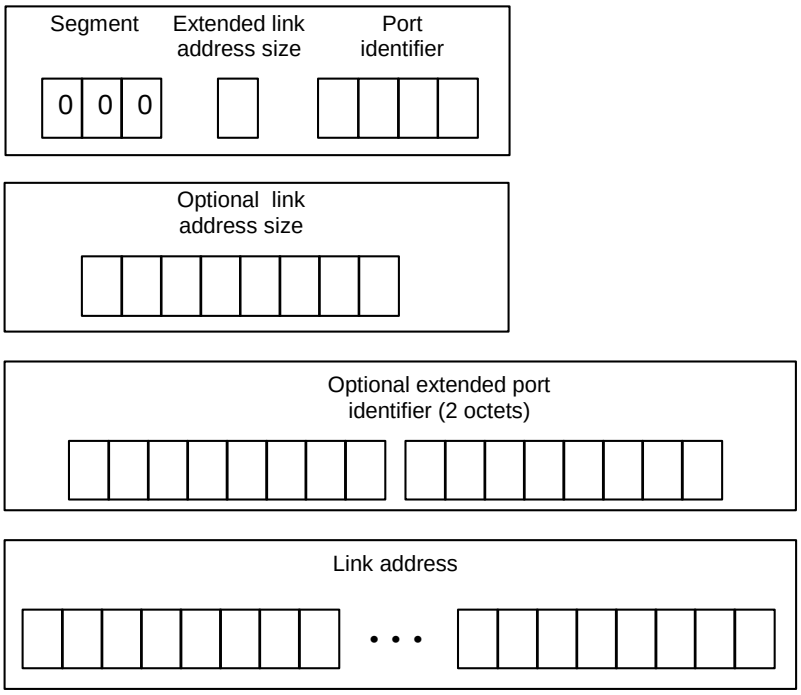
Bits 5 to 7 of the first octet shall be zero to indicate the port segment type.

Bit 4, the extended link address size bit, shall be set to 0 if the link address is one octet. Bit 4 shall be set to 1 if the link address is larger than one octet. If the link address is larger than one octet, its size in octets shall be in the second octet of the port segment.

Bits 0 to 3, the port identifier, shall indicate through which port to leave the node. The port identifier shall specify a port number or an escape to an extended port identifier when the module can support more than 14 ports. Port number 0 shall be reserved. Port number 1 shall only be used to represent a back-plane port. If the port identifier is 15, then a 16-bit field, called the extended port identifier, shall be the next part of the port segment (following the optional link address size if present); otherwise, the value of the port identifier shall be the port number.

The port number shall be followed by a link address whose format depends on the type of network to which the port identifier refers. If the link address is greater than one octet, then it shall be padded so that the entire port segment is an even number of octets. The pad octet shall be set to 0 and shall not be included in the link address size.

NOTE 1 For the common port types, the link address is a single octet. Other port types, such as TCP/IP encapsulation, can use a larger link address (see 4.3 and Clause 11).



NOTE The bit representation is from high bit to low bit, left to right. The octet representation is from low octet to high octet, top to bottom and left to right.

Figure 11 – Port segment

The Extended Port Identifier format of the Port Segment shall only be used when there are more than 14 ports possible on the device. The Port Segment shall always be packed into the smallest Port Segment format possible with respect to the optional fields. Examples of possible port segments are as shown in Table 156.

Table 156 – Possible port segment examples

Port segment contents	Notes
[02][06]	Segment type = port segment, port number = 2, link address = 6
[0F][12][00][01]	Segment type = port segment. Port identifier is 15, indicating the port number is specified in the next 16 bit field [12][00] (18 decimal). Link address = 1
[15][0F][31][33][30][2E] [31][35][31][2E][31][33] [37][2E][31][30][35][00]	Segment type = port segment. Multi-octet address for TCP Port 5, link address 130.151.137.105 (IP address). The address is defined as a character array, length of 15 octets. The last octet in the segment is a pad octet

The link address portion of a TCP/IP connection path segment shall be encoded within the port segment as a string of ASCII characters. The string shall be one of the following forms:

- IP address in dot notation, for example “130.151.132.55” (see IETF RFC 1117 for the format of IP addresses);
- IP address in dot notation, followed by a “:” separator, followed by the TCP port number to be used at the specified IP address;
- host name, for example “plc.type2.org”. The host name shall be resolved via a DNS request to a name server (see IETF RFC 1035 for information on host names and name resolution);
- host name, followed by a “:” separator, followed by the TCP port number to be used at the specified host.

The port number shall be represented in either hex or decimal. Hex shall be indicated by a leading “0x”. When a port number is specified, it shall be used rather than the standard port number used for the encapsulation protocol (0xAF12). Only port 0xAF12 is guaranteed to be available in a CP 2/2 compliant device.

Other TCP port numbers may be implemented; however, this specification does not provide a mechanism to determine which TCP port numbers are supported by a device. The use of other TCP port numbers is therefore discouraged.

NOTE 2 The guaranteed TCP port number, 0xAF12, has been reserved with the Internet Assigned Numbers Authority (IANA) for use by the encapsulation protocol.

Since port segments shall be word-aligned, a pad octet may be required at the end of the string. The pad octet shall be 0x00, and shall not be counted in the Optional Address Size field of the port segment.

NOTE 3 Examples of port segments for TCP/IP are shown in Table 157 below.

Table 157 – TCP/IP link address examples

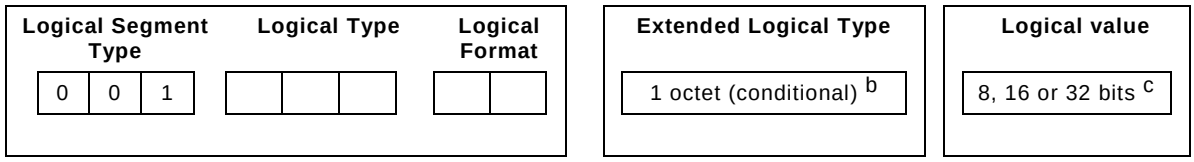
Port segment	IP address	Notes
[12][0D] [31][33][30][2E] [31][35][31][2E] [31][33][32][2E][31][00]	130.151.132.1	Multi-octet address for port 2, 13 octet string plus a pad octet
[13][0C] [70][6C][63][2E] [74][79][70][32][2E] [6F][72][67]	plc.typ2.org	Multi-octet address for port 3, 12 octet string, no pad octet
[16][15] [31][33][30][2E] [31][35][31][2E] [31][33][32][2E] [35][35][3A] [30][78][33][32][31][30][00]	130.151.132.55:0x3210	Multi-octet address for port 6, 21 octet string plus a pad octet
[15][11] [70][6C][63][2E] [74][79][70][32][2E] [6F][72][67][3A] [39][38][37][36][00]	plc.typ2.org:9876	Multi-octet address port 5, 17 octet string plus a pad octet

4.1.9.4 Logical segment

4.1.9.4.1 Logical segment structure

The logical segment selects a particular addressable entity within a device (for example, object class, object instance, and object attribute). When the logical segment is included within a Packed Path, the Logical Value shall be appended to the segment type octet with no pad in between.

Figure 12 specifies the encoding of a logical segment.



Class ID	0	0	0	0	0	8-bit logical value
Instance ID	0	0	1	0	1	16-bit logical value
Member ID	0	1	0	1	0	32-bit logical value
Connection Point	0	1	1	1	1	Reserved for future use
Attribute ID	1	0	0			
Special [a]	1	0	1			
Service ID [a]	1	1	0			
Extended Logical	1	1	1			

[a] The Special and Service ID Logical Types do not use the logical addressing definition for the Logical Format.

[b] Required when Logical Type is Extended Logical (1 1 1), not present otherwise. The Extended Logical Type values are defined in Table 158.

[c] Depends on value of Logical Format.

Figure 12 – Logical segment encoding

The Extended Logical Types are specified in Table 158.

Table 158 – Extended Logical Type

Value	Description
0	Reserved
1	Array Index
2	Indirect Array Index
3	Bit Index
4	Indirect Bit Index
5	Structure Member Number
6	Structure Member Handle
7 to 255	Reserved

The 8-bit and 16-bit logical address formats are allowed for use with all Logical Types.

The 32-bit logical address format is only allowed for the logical Instance ID, Connection Point, Array Index, Bit Index, Structure Member Number and Structure Member Handle types. It is not allowed for any other Logical Type (reserved for future use).

The Class ID specifies a type of object within a device. The Instance ID is an integer assigned to an object when it is created. Instance 0, called the class instance, specifies the class itself. The attributes of an object are an enumerated list of externally visible values. The attribute ID specifies which attribute of an object is selected. Complex attributes are composed of members. The member ID specifies which member of the attribute.

When an extended logical segment is included within a Padded Path and the Logical Format is 8 bit, a pad octet shall be added after the Logical Value (the 16-bit and 32-bit formats are identical to the Packed Path) and shall be set to 0. For all other logical segments, when included within a Padded Path, the 16-bit and 32-bit logical formats shall have a pad inserted

between the segment type octet and the Logical Value (the 8-bit format is identical to the Packed Path). The pad octet shall be set to zero.

When an Extended Logic Segment is specified the Extended Logical Segment Value octet shall be included between the Segment Type octet and the Logical Value.

An Array Index Extended Logical Segment is used to specify the 0-based index into the preceding item in the EPATH which shall be an array.

An Indirect Array Index Extended Logical Segment is used to specify the start of a nested application path, which resolves to a 0-based index into the preceding item in the EPATH which shall be an array. The Indirect Array Index Logical Segment Logical Value is the size of the nested application path, in 16-bit words.

A Bit Index Extended Logical Segment is used to specify the 0-based bit number into the preceding item in the EPATH (there is no restriction on type of the preceding item).

An Indirect Bit Index Extended Logical Segment is used to specify the start of a nested application path, which resolves to a 0-based bit number into the preceding item in the EPATH (there is no restriction on type of the preceding item). The Indirect Bit Index Logical Segment Logical Value is the size of the nested application path, in 16-bit words.

A Structure Member Number Extended Logical Segment is used to specify the 1-based member number into the preceding item in the EPATH which shall be a structure. The member number specifies the structure member, where the structure members are numbered starting with 1.

A Structure Member Handle Extended Logical Segment is used to specify the member handle into the preceding item in the EPATH which shall be a structure. The handle is a value that uniquely identifies a member and is provided by the target device when reporting constructed data type information using the Formal Encoding with Handles specified in 5.2.3.2.5.

The Connection Point Logical Type provides additional addressing capabilities beyond the standard Class ID/Instance ID/Attribute ID/Member ID object address. Object classes shall define when and how this addressing component is utilized (for example, a sub-instance of an Assembly object).

NOTE IEC 61158-5-2 describes the object model including the definition of class, instance, attribute, connection point, and members of a library of common objects.

The Service ID Logical Type has the following definition for the Logical Format:

- 0 0 8-Bit Service ID Segment (0x38)
- 0 1 Reserved for future use (0x39)
- 1 0 Reserved for future use (0x3A)
- 1 1 Reserved for future use (0x3B)

The Special Logical Type has the following definition for the Logical Format:

- 0 0 Electronic Key Segment (0x34)
- 0 1 Reserved for future use (0x35)
- 1 0 Reserved for future use (0x36)
- 1 1 Reserved for future use (0x37)

4.1.9.4.2 Electronic Key Segment

The electronic key segment shall be used to verify/identify a device. The key may be included as the first logical segment in a connection path and shall be checked by the Message Router of the receiving node against the values contained in instance 1 of its Identity object before any additional address checks are made. On a multi-hop message, sent to a remote link via intermediate routers, multiple key segments may appear. The format of the electronic key segment shall be as shown in Table 159.

Table 159 – Electronic key segment format

Parameter	Format	Value
segment_type	USINT	always 0x34
electronic_key_type	USINT	always 0x04
vendor_ID	UINT	these correspond to the values in instance 1 of the identity object
product_type	UINT	
product_code	UINT	
major_revision : 7	USINT	
compatible_match: 1	USINT	
minor_revision	USINT	

The `segment_type` for a key segment shall be 0x34. The `electronic_key_type` shall determine the format of a specific key segment and shall be set to 0x04. All other values for the `electronic_key_type` shall be reserved.

The `vendor_ID` shall specify the device vendor, or zero if no specific vendor is required.

The `product_type` shall specify a class of products such as digital input or analogue outputs. The `product_type` shall be ignored when it is set to zero. The `product_code` shall be vendor specific with each vendor having a unique code. The `product_code` shall be ignored when set to zero.

The `major_revision` and `compatible_match` fields shall be contained in the least significant 7 bits and most significant bit of the same octet, respectively.

The `major_revision` shall specify the functionality level of a device. The `major_revision` shall be ignored when set to zero.

The `minor_revision` shall specify the revision of a device that does not effect network-visible behaviour. A value of zero shall specify that the originator of the request accepts any minor revision.

The `compatible_match` shall modify the key matching function.

If the bit is clear (= 0), then any non-zero `vendor_ID`, `product_type`, `product_code`, `major_revision`, and `minor_revision` shall match exactly.

If the bit is set (= 1), the device may accept the key for any device which it can emulate. The level of emulation shall be product specific. When this bit is set (1), 0 is an invalid value for `vendor_ID`, `product_code` and `major_revision`. If the `vendor_ID`, `product_code` or `major_revision` value is 0, error code 0x04 (Path segment error) shall be returned. The `minor_revision` value shall be ignored, any value may be specified. There is no restriction

on the product_type value. A product_type value of 0 indicates the device shall be compatible with the Generic Device type.

NOTE 2 IEC 61158-5-2 describes the Identity object and the rules for updating the revision fields.

4.1.9.4.3 Logical segments examples

Table 160 shows examples of logical segments.

Table 160 – Logical segments examples

First octet	Logical segment name	Padded format examples (as transmitted)	Packed format examples (as transmitted)	Notes
0x20	8-bit class	[20][04]	[20][04]	class 0x04 (Assembly object)
0x21	16-bit class	[21][00][34][12]	[21][34][12]	class 0x1234
0x24	8-bit instance	[24][04]	[24][04]	instance 0x04
0x25	16-bit instance	[25][00][34][12]	[25][34][12]	instance 0x1234
0x28	8-bit member	[28][04]	[28][04]	member 0x04
0x29	16-bit member	[29][00][34][12]	[29][34][12]	member 0x1234
0x2A	32-bit member			
0x2C	8-bit connection point	[2C][04]	[2C][04]	connection point 0x04
0x2D	16-bit connection point	[2D][00][34][12]	[2D][34][12]	connection point 0x1234
0x30	8-bit attribute	[30][04]	[30][04]	attribute 0x04
0x31	16-bit attribute	[31][00][34][12]	[31][34][12]	attribute 0x1234
0x34	electronic key	[34][04] [12][00] [EF][BE] [0E][0B] [83] [01]	[34][04] [12][00] [EF][BE] [0E][0B] [83] [01]	vendor ID = 0x0012 product type = 0xBEEF product code = 0x0B0E major revision = 0x03 minor revision = 0x01 accept compatible keys = yes

4.1.9.5 Network segment

4.1.9.5.1 Network segment structure

The segment type (first octet) of a network segment shall be in the range 0x40 through 0x5F as shown in Table 161 (the most significant bits shall be 010). The network segment shall be used to specify network parameters which may be required by a node to transmit a message across a network. The network segment shall immediately precede the port segment of the device to which it applies. In other words, the network segment shall be the first item in the path that the device receives.

Table 161 – Network segments

First octet	Network segment name
0x40	reserved
0x41	schedule segment
0x42	fixed tag segment
0x43	production inhibit time segment
0x44 to 0x4F	reserved
0x050	safety segment
0x051 to 0x05E	reserved
0x5F	extended network segment

4.1.9.5.2 Schedule Segment

The segment type of the schedule network segment shall be 0x41. The schedule network segment shall specify

- a multiplier that, when multiplied by the NUT, gives the CM_API (actual packet interval) of the scheduled transport;
- a phase that determines on which values of the `Moderator.interval_count` to transmit.

NOTE 1 The data-link layer defines the Moderator.interval_count, see IEC 61158-4-2.

This segment shall be included in the path to a device so that each intermediary node can schedule link transmit time for subsequent scheduled traffic. The multiplier shall be one of 1, 2, 4, 8, 16, 32, 64 or 128. The phase shall be in the range 0 through (multiplier – 1). The second octet of the schedule network segment shall encode both the multiple and phase by adding them together.

NOTE 2 If a transport produces every 64th NUT starting on `interval_count` = 17, then the encoded value is 17 + 64 = 81.

An encoded value of zero shall specify that the transport has not yet been given permission to use the scheduled priority. If a node receives a request for a scheduled connection in which either no schedule network segment exists, or the schedule network segment exists but the encoded value is 0, the node shall return general status = 0x01, and extended status = 0x0317.

NOTE 3 A connection originator is given permission to use the scheduled priority through the Scheduling object (IEC 61158-5-2).

For intermediate nodes, if only one of the incoming or outgoing ports connects to a CP 2/1 subnet, then a single schedule segment is required if the priority is scheduled. The schedule segment applies to the port connected to the CP 2/1 subnet.

For intermediate nodes, if both the incoming and outgoing ports connect to CP 2/1 subnets, then two schedule segments are required if the priority is scheduled. The first schedule segment applies to the incoming port and the second schedule segment applies to the outgoing port.

For target nodes, a single schedule segment is required if the incoming port connects to a CP 2/1 subnet and the priority is scheduled.

4.1.9.5.3 Fixed Tag Segment

The segment type of the fixed tag network segment shall be 0x42. The fixed tag network segment shall specify the fixed tag which is to be used when sending an unconnected message. This segment subtype shall precede the port segment within the path. If the fixed tag segment is not present, then the default fixed tag shall be used.

NOTE The fixed tag segment can be used within the path of the Unconnected_Send service (see 4.1.5.6) of the Connection Manager to send requests to the Keeper object via the Management UCMM (see IEC 61158-5-2).

4.1.9.5.4 Production Inhibit Time Segment

The segment type of the production inhibit time segment shall be 0x43. The second octet shall specify the production inhibit time, which is the minimum time, in milliseconds, between successive transmissions of connected data for the specified connection (range of 1 ms to 255 ms. If the production inhibit time is 0, there shall be no limit on how fast data may be sent on the connection.

EXAMPLE If a production inhibit time of 10 ms is specified, new data shall be sent no sooner than 10 ms after the previous data.

When the production inhibit time segment is used, the RPI shall determine when to re-send data. If the production inhibit time segment is omitted during establishment of a connection, a default production inhibit time of 1/4 the RPI shall be used. The RPI shall be greater than or equal to the production inhibit time. If the RPI is smaller than the production inhibit time, the Forward_Open response shall be returned with one of the following errors:

- RPI is smaller than the production inhibit time (status 0x01, extended status 0x11B) – recommended;
- RPI not supported (status 0x01, extended status 0x0111) – deprecated.

The production inhibit time network segment only applies to the target device.

NOTE The method for transmitting APDUs over Ethernet-TCP/IP is specified in 4.3 and Clause 11.

4.1.9.5.5 Safety Segment

See IEC 61784-3-2 for format.

4.1.9.5.6 Extended Network Segment

The extended network segment allows for the definition of additional network segment subtypes. The first word of data is the extended network segment subtype. The structure of the extended network segment is shown in Figure 13.

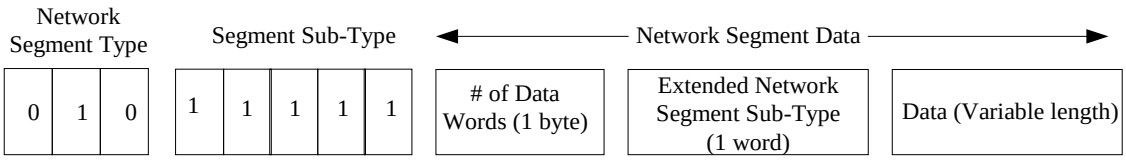


Figure 13 – Extended network segment

Each extended subtype defines the data that follows. The extended subtypes are enumerated in Table 162.

Table 162 – Extended subtype definitions

Extended Subtype Value	Extended Subtype Definition
0 to 32 767	Reserved for future use
32 768 to 65 535	Vendor specific [a]

[a] Vendor specific network segments shall only be targeted to the last device in the Connection_Path parameter of the Connection Manager object services.

4.1.9.6 Symbolic segment

The range of segment types reserved for symbolic segments shall be 0x60 through 0x7F.

The symbolic segment contains an International String symbol which shall be interpreted by the device. Its format is specified in Figure 14.

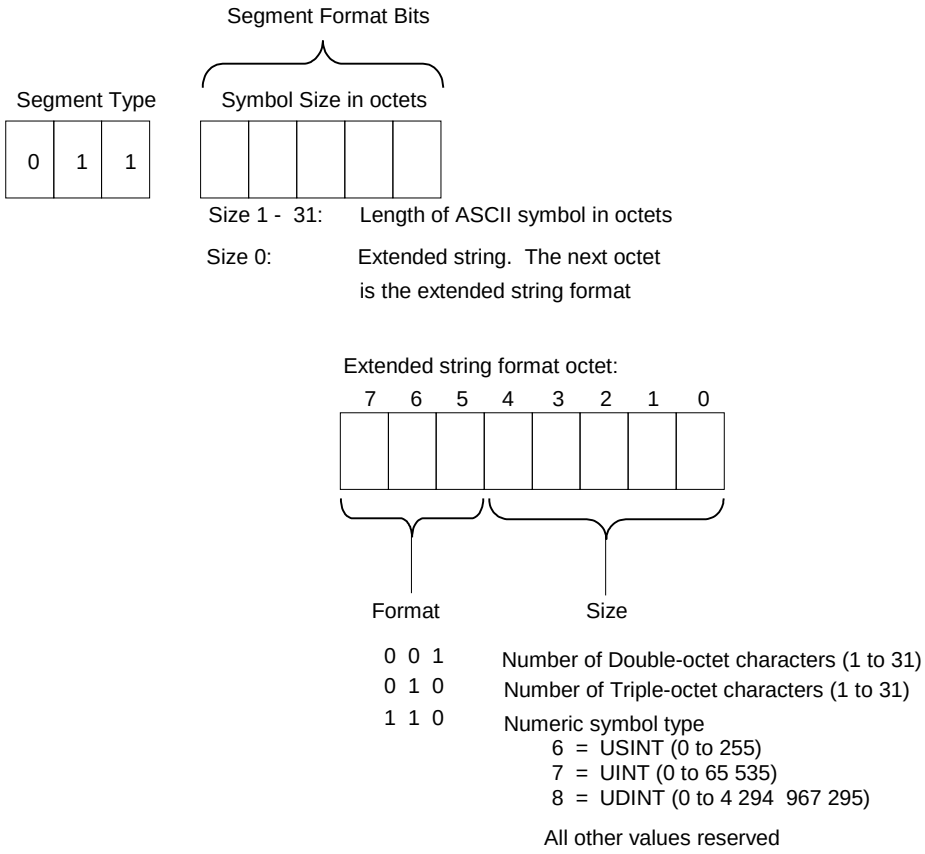


Figure 14 – Symbolic segment encoding

Character symbols can contain a maximum of 31 characters.

If the symbol is ASCII, double-octet or triple-octet, when comparing segments, the symbol portion shall be handled in a case insensitive manner. For instance “LS101” and “ls101” shall be considered equal.

Numeric symbols can be 8, 16, or 32 bits.

When a symbolic segment is used in a Padded EPATH, if the length of the segment is odd a pad octet of 0 shall be added at the end of the segment. The size shall not include the pad octet.

Table 163 shows examples of symbolic segments (values are hexadecimal).

Table 163 – Symbolic segment examples

Symbolic Segment	Notes
[65][LS101]	ASCII Symbol. This could refer to a bit in a data structure
[67][Line_23]	ASCII Symbol. This could refer to a controller in a slot
[68][Wire_off]	ASCII Symbol. This could refer to a bit in a diagnostic structure
[60][22][1234][2345]	Japanese symbol
[60][C7][1234]	16 bit Numeric Symbol
[60][C8][12345678]	32 bit Numeric Symbol

4.1.9.7 Data segment

4.1.9.7.1 Data segment purpose and structure

The data segment provides a mechanism for delivering data to an application. This may occur during connection establishment, or at any other time as defined by the application.

The segment type (first octet) of a data segment shall be in the range 0x80 through 0x9F (the three most significant bits shall be 100). This standard only defines the format of the simple data segment with a segment type of 0x80; and the ANSI extended symbol segment (0x91). All other data segment types shall be reserved (0x81 through0x90, 0x92 through 0x9F).

4.1.9.7.2 Simple Data Segment

The segment type of the simple data segment shall be 0x80. The second octet shall represent the number of 16-bit words of variable length data (up to 255). Following the second octet shall be the variable length data as shown in Table 164. A path may contain more than one simple data segment.

Table 164 – Data segment

Parameter	Format	Value
segment_type	USINT	always 0x80
segment_size	USINT	size of the data[] array in 16-bit words
data[]	UINT	

The simple data segment contains data values such as parameters for the target application. The data can be configuration information for an object, additional parameters necessary for the object, or any other set of object specific information. The data segment shall not be interpreted by any other device in the path, so only the originating and target applications need agree on its contents.

4.1.9.7.3 ANSI Extended Symbol Segment

The segment type of the ANSI extended symbol segment shall be 0x91. The second octet shall represent the number of characters (8-bit) in the symbol. Following the second octet shall be the variable length symbol as shown in Table 165.

The character portion of the ANSI extended symbol segment shall be handled in a case insensitive manner. For instance “start1” and “Start1” shall be considered equal.

Table 165 – ANSI_Extended_Symbol segment

Parameter	Format	Value
segment_type	USINT	always 0x91
symbol_size	USINT	size of the symbol[] array
symbol[]	USINT	
pad	USINT	only present if symbol_size is odd, always set to zero

The `symbol_size` shall be the size of the `symbol[]` array in octets and shall not be zero. The `symbol_size` shall not count the `pad` octet if it is included.

4.1.9.8 Segment definition hierarchy

In general, the definition of any rules related to the use of segments is defined within the Object Class and/or Device Profile. When symbolic and/or logical segments are used to construct an application path, the Backus-Naur Form definition is:

application_path::= CHOICE {*Symbolic_application_path*, *Class_application_path*}

Symbolic_application_path::= *symbolic_segment* [Connection_Point] [*member_specification*] [*bit_specification*]

symbolic_segment::= CHOICE {Symbolic_Segment, ANSI_Extended_Symbol_Segment}

Class_application_path::= Class_ID [*item_specification*]

item_specification::= CHOICE {*attribute_specification*, *connection_point_specification*}

attribute_specification::= Instance ID [[Attribute ID] [*member_specification*] [*bit_specification*]]

connection_point_specification::= [Instance ID] Connection Point [*member_specification*] [*bit_specification*]

member_specification::= CHOICE {Member_ID, *extended_member_specification*}

extended_member_specification::= SEQUENCE of {Array_Index, *indirect_array_specification*, Structure_Member_Number, Structure_Member_Handle}

indirect_array_specification::= Indirect_Array_Index *application_path*

bit_specification::= CHOICE {Bit_Index, *indirect_index_bit_specification*}

indirect_bit_specification::= Indirect_Bit_Index *application_path*

A *Symbolic_application_path* shall resolve, at run time, to a *Class_application_path*.

An Indirect_Array_Index and Indirect_Bit_Index *application_path* shall evaluate to a positive integer.

The depth within the application path need only proceed to the degree required by its application.

EXAMPLE

The following are examples of valid application paths.

Class ID,

Class ID, Instance ID

Class ID, Instance ID, Attribute ID

Class ID, Instance ID, Attribute ID, Member ID

Class ID, Connection Point

Class ID, Connection Point, Member ID

Class ID, Instance ID, Connection Point (see constraint a) below)

Class ID, Instance ID, Connection Point, Member ID (see constraint a) below)

Symbolic ID

Symbolic ID, Member ID

Symbolic ID, Connection Point (see constraint b) below)

Symbolic ID, Connection Point, Member ID (see constraint b) below)

Symbolic ID, Array Index, Structure Member Number, Array Index, Bit Index

Symbolic ID, Indirect_Array_Index, [Class_ID, Instance_ID, Attribute_ID], Structure_Member_Number, Array_Index, Bit_Index

The following constraints apply:

c) Instance ID and Connection Point for Class ID 4 (Assembly Object) shall not be used together, since for the Assembly object they are defined to be the same.

d) If the Symbolic ID resolves to a Class ID of 4 (Assembly Object) and Instance ID then Symbolic ID and Connection Point shall not be used together, since for the Assembly object Instance and Connection Point are defined to be the same.

4.1.9.9 Encoded path compression rules

When multiple encoded paths are concatenated, the delineation between paths is where a segment at a higher level in the hierarchy is encountered. Multiple encoded paths may be compacted when each path shares the same values at the higher levels in the hierarchy. Extended Logical segments shall not be used in compressed paths. When a segment is encountered which is at the same or higher level but not at the top level in the hierarchy, the preceding higher levels are used for that next encoded path.

The examples below show multiple encoded paths in the full and compacted representations.

EXAMPLE 1

Full: Class A, Instance A, Attribute A, Class A, Instance A, Attribute B

Compact: Class A, Instance A, Attribute A, Attribute B

EXAMPLE 2

Full: Class A, Instance A, Attribute A, Class A, Instance B, Attribute A

Compact: Class A, Instance A, Attribute A, Instance B, Attribute A

The following special compact format is also defined, but only when the class is not the assembly object:

Full: Class A, Instance B, Class A, Instance B, Connection Point C, Class A, Instance B, Connection Point D, Data Segment

Compact: Class A, Instance B, Connection Point C, Connection Point D, Data Segment

When a connection path includes a configuration or I/O path consisting of an instance of the Assembly object (class code 4) without an attribute, the data attribute (attribute 3) is assumed.

EXAMPLE

An encoded path with the contents 20 04 24 xx 24 yy 24 zz followed by a data segment is decoded as follows:

20 04 24 xx 30 03 configuration path

20 04 24 yy 30 03 consuming I/O path

20 04 24 zz 30 03 producing I/O path

4.1.10 Class, attribute and service codes

4.1.10.1 Code ranges

4.1.10.1.1 Defined ranges

There shall be three categories for address ranges of Class IDs, Attribute IDs and Service Codes as specified in Table 166.

Table 166 – Addressing categories

This category	Refers to
Open	A value which has the same meaning for all implementers. All object classes and services defined in IEC 61158-5-2 fall under this category.
Vendor-specific	A range of values specific to the vendor of a device. These are used by vendors to extend their devices beyond the available *Open* options. A vendor internally manages the use of values within this range. Applies to object classes, attributes, and services.
Object-class specific	A range of values whose meaning is defined by an object class. This range applies to Service Code definitions.
NOTE Open values are assigned by ODVA, Inc.	

The Class ID, Attribute ID and Service code values shall be as defined in Table 167 through Table 169.

4.1.10.1.2 Class code ID ranges

The Class ID values shall be as shown in Table 167. Class Code = 0x00 is reserved and shall not be used.

Table 167 – Class code ID ranges

Range (hex)	Range (decimal)	Meaning	Quantity
00 to 63	01 to 99	Open	99
64 to C7	100 to 199	Vendor Specific	100
C8 to EF	200 to 239	Reserved	40
F0 to 2FF	240 to 767	Open	528
300 to 4FF	768 to 1 279	Vendor Specific	512
500 to FFFF	1 280 to 65 535	Reserved	64 256
NOTE Reserved ranges may be further defined by ODVA, Inc.			

4.1.10.1.3 Attribute ID ranges

The Attribute ID values shall be as shown in Table 168. Attribute ID = 0x00 is reserved and shall not be used.

Table 168 – Attribute ID ranges

Range (hex)	Range (decimal)	Meaning	Quantity
00 to 63	00 to 99	Open	100
64 to C7	100 to 199	Vendor Specific	100
C8 to FF	200 to 255	Reserved	56
100 to 2FF	240 to 767	Open	512
300 to 4FF	768 to 1 279	Vendor Specific	512
500 to 8FF	1 280 to 2 303	Open	1 024
900 to CFF	2 304 to 3 327	Vendor Specific	1 024
D00 to FFFF	3 328 to 65 535	Reserved	62 208
NOTE Reserved ranges may be further defined by ODVA, Inc.			

4.1.10.1.4 Service code ranges

The **Service code** shall be a unique hexadecimal value assigned to each service. The Service Code values shall be as shown in Table 169. Service Code = 0x00 is reserved and shall not be used. The object-specific service codes shall be unique only within the class which define them.

Table 169 – Service code ranges

Range (hex)	Range (decimal)	Meaning	Quantity
00	00	Reserved	1
01 to 31	01 to 49	Open. These are referred to as *Common Services*. They are defined in IEC 61158-5-2	49
32 to 4A	50 to 74	Vendor Specific	25
4B to 63	75 to 99	Object Class Specific	25
64 to 7F	100 to 127	Reserved	28
80 to FF	128 to 255	Reserved for response messages	128
NOTE Reserved range may be further defined by ODVA, Inc.			

4.1.10.2 Code definitions

4.1.10.2.1 Communication object classes

Table 170 defines the class codes for the object classes. All other class codes listed within the “open” range and not listed in Table 170 shall be reserved. The requirements for these objects are defined in IEC 61158-5-2.

Table 170 – Class codes

Class code (hex)	Class code (decimal)	Object name
0x01	1	Identity object
0x02	2	Message Router object
0x03	3	DeviceNet object[a]
0x04	4	Assembly object
0x05	5	Connection object
0x06	6	Connection Manager object
0x0F	15	Parameter object
0x2B	43	Acknowledge Handler object
0x39	57	Safety Supervisor object[b]
0x3A	58	Safety Validator object[b]
0x42	66	Motion Device Axis object[c]
0x43	67	Time Sync object
0x47	71	DLR object[a]
0x48	72	QoS object[a]
0x4C	76	SERCOS III Link object[d]
0xF0	240	ControlNet object[a]
0xF1	241	Keeper object[a]
0xF2	242	Scheduling object[a]
0xF3	243	Connection Configuration object[a]
0xF4	244	Port object[a]
0xF5	245	TCP/IP Interface object[a]
0xF6	246	Ethernet Link object[a]

[a] This object class is part of system management.

[b] This object class is specified in IEC 61784-3-2.

[c] This object class is specified in IEC 61800-7-202.

[d] This object class will be specified in IEC 61784-3-2 Edition 3.0 (currently under preparation).

4.1.10.2.2 Predefined class attributes

Seven predefined Class Attribute IDs shall be reserved, and these predefined/reserved class attributes shall have the definitions listed in Table 171. Because these attributes are reserved, class Attribute ID numbers 1 through 7 shall always be reserved. Therefore, if a class attribute is added to an object class specification, it shall start with Attribute ID #8.

Table 171 – Reserved class attributes for all object class definitions

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	The starting value assigned to this attribute is one (1). If updates that require an increase in this value are made, then the value of this attribute increases by one (1).
2	Max Instance	UINT or UDINT	The class definition shall define the data type used.
3	Number of Instances	UINT or UDINT	The class definition shall define the data type used.
4	Optional attribute list	STRUCT of	

Attribute ID	Name	Data type	Semantics of values
	Number of attributes	UINT	
	Optional attributes	ARRAY of UINT	
5	Optional service list	STRUCT of	
	Number services	UINT	
	Optional services	ARRAY of UINT	
6	Maximum ID Number Class Attributes	UINT	
7	Maximum ID Number Instance Attributes	UINT	

4.1.10.2.3 OM_Service codes

Codes of the common services for the deterministic control network objects shall be as shown in Table 172.

NOTE Table 172 lists the codes and names of the common services while IEC 61158-5-2 provides a general description of each service.

Table 172 – Common services list

Common service code	Common service name
0x00	reserved
0x01	Get_Attribute_All
0x02	Set_Attribute_All
0x03	Get_Attribute_List
0x04	Set_Attribute_List
0x05	Reset
0x06	Start
0x07	Stop
0x08	Create
0x09	Delete
0x0A to 0x0C	reserved
0x0D	Apply_Attributes
0x0E	Get_Attribute_Single
0x0F	reserved
0x10	Set_Attribute_Single
0x11	Find_Next_Object_Instance
0x12 to 0x14	reserved
0x15	Restore
0x16	Save
0x17	NOP (No operation)
0x18	Get_Member
0x19	Set_Member
0x1A	Insert_Member
0x1B	Remove_Member
0x1C	Group_Sync
0x1D to 0x31	reserved

Codes of the object specific services for the Message Router object shall be as shown in Table 173.

Table 173 – Message Router object specific services list

Object specific service code	Service name
0x4B	Symbolic_Translation

Codes of the object specific services for the Acknowledge Handler object shall be as shown in Table 174.

Table 174 – Acknowledge Handler object specific services list

Object specific service code	Service name
0x4B	Add_AckData_Path
0x4C	Remove_AckData_Path

Code of the object specific service for the Parameter object shall be as shown in Table 175.

Table 175 – Parameter object specific services list

Object specific service code	Service name
0x4B	Get_Enum_String

4.1.10.2.4 CM_Service codes

Codes of the services specific to the Connection Manager shall be as shown in Table 176.

NOTE Table 176 lists the codes and names of the services while IEC 61158-5-2 provides a general description of each service.

Table 176 – Services specific to Connection Manager

Service code	Service name
0x4E	Forward_Close
0x52	Unconnected_Send
0x54	Forward_Open
0x56	Get_Connection_Data
0x57	Search_Connection_Data
0x5A	Get_Connection_Owner
0x5B	Large_Forward_Open

4.1.10.2.5 CO_Service codes

Codes of the services specific to the Connection object shall be as shown in Table 177.

NOTE Table 177 lists the codes and names of the services while IEC 61158-5-2 provides a general description of each service.

Table 177 – Services specific to Connection object

Service code	Service name
0x4B	Connection_Bind
0x4C	Producing_Application_Lookup
0x4E	SafetyClose
0x54	SafetyOpen

4.1.10.3 Device types

In order to allow interoperability and interchangeability, a device type shall be used to identify similar devices which

- exhibit the same behaviour;
- produce and/or consume the same set of data;
- contain the same set of configurable attributes.

The formal definition of this information is known as a Device Profile: all devices with the same device type number shall meet the requirements specified in the Device Profile for that device type.

"Device Type" is a required instance attribute of the Identity object as described in IEC 61158-5-2.

Device types shall be either publicly defined or vendor specific. Table 178 reveals the numbering scheme to be used for device type numbering.

Table 178 – Device type numbering

Range (hex)	Type	Quantity
00 to 63	Publicly Defined – Reserved	100
64 to C7	Vendor Specific	100
C8 to FF	Reserved	56
100 to 2FF	Publicly Defined – Reserved	512
300 to 4FF	Vendor Specific	512
500 to FFFF	Reserved	64 256

All publicly defined device types shall have the same meaning for all implementers and are reserved. The Generic Device type (type number = 0x00) shall define a device that does not fit into any of the defined device types.

NOTE The actual definition of publicly defined device types and corresponding Device Profiles is outside the scope of this standard. They are defined by ODVA, Inc.

Vendor specific device types may be developed and vendors need not publish them. Each vendor shall maintain their own vendor-specific device types and corresponding number allocation.

4.1.11 Error codes

4.1.11.1 CM errors

The error codes are returned with the response to a Connection Manager Service Request which resulted in an error. These error codes shall be used to help diagnose the problem with a Service Request. The error code shall be split into an 8 bit general status and one or more

16 bit words of extended status. Unless specified otherwise, only the first word of extended status shall be required. Additional words of extended status may be used to specify additional device specific information. All devices which originate messages shall be able to handle multiple words of extended status.

Table 179 provides a summary of the available error codes.

Only the type of node indicated in the "Detected by" column shall return the general status/extended status code. The "Detected by" column values are Originator, Router and/or Target. When detected by Router or Target devices these error codes will be contained in the response packet. When detected by Originator these error codes may be reported back to the application via an object interface (e.g. Connection Configuration object) or other internal interface.

Table 179 – Connection Manager service request error codes

General Status	Extended Status	Detected by	Explanation and description
0x00			Service completed successfully.
0x01	0x0000 to 0x00FF		Obsolete
0x01	0x0100	Router Target	CONNECTION IN USE OR DUPLICATE FORWARD OPEN This extended status code shall be returned when an originator is trying to make a connection to a target with which the originator may have already established a connection (Non-null/matching Forward_Open – see 4.1.5.3.2.2).
0x01	0x0101 to 0x0102		Reserved
0x01	0x0103	Target	TRANSPORT CLASS AND TRIGGER COMBINATION NOT SUPPORTED A transport class and trigger combination has been specified which is not supported by the target application.
0x01	0x0104 to 0x0105		Reserved
0x01	0x0106	Target	OWNERSHIP CONFLICT The connection cannot be established since another connection has exclusively allocated some of the resources required for this connection. An example of this would be that only one exclusive owner connection can control an output point on an I/O Module. If a second exclusive owner connection (or redundant owner connection) is attempted, this error shall be returned.
0x01	0x0107	Target	TARGET CONNECTION NOT FOUND This extended status code shall be returned in response to the Forward_Close request, when the connection that is to be closed is not found at the target node. Routers shall not generate this extended status code. If the specified connection is not found at the intermediate node, the close request shall still be forwarded using the path specified in the Forward_Close request.
0x01	0x0108	Router Target	INVALID NETWORK CONNECTION PARAMETER This extended status code shall be returned as the result of specifying a connection type, connection priority, redundant owner or fixed/variable that is not supported by the device. NOTE This extended status code is "deprecated". It is highly recommended that 0x011F, 0x0120, 0x0121, 0x0122, 0x0123, 0x0124, 0x0125 or 0x0132 be used instead.
0x01	0x0109	Router Target	INVALID CONNECTION SIZE This extended status code is returned when the target or router does not support the specified connection size. This could occur at a target because the size does not match the required size for a fixed size connection. It could occur at a router if the requested size is too large for the specified network. An additional status word may follow indicating the maximum connection size supported by the responding node. The additional status word is required when issued in response to the Large_Forward_Open. NOTE This extended status code is "deprecated". It is highly recommended that 0x0126, 0x0127, or 0x0128 be used instead.
0x01	0x010A to 0x010F		Reserved
0x01	0x0110	Target	TARGET FOR CONNECTION NOT CONFIGURED This extended status code shall be returned when a connection is requested to a target application that has not been configured and the connection request does not contain a data segment for configuration (see 4.1.9.7).

<table>
<tr><th>General Status</th><th>Extended Status</th><th>Detected by</th><th colspan="3">Explanation and description</th></tr>
<tr><td>0x01</td><td>0x0111</td><td>Router
Target</td><td colspan="3">RPI NOT SUPPORTED.
This extended status code shall be returned if the device can not support the requested O⇒T or T⇒O RPI. This extended status code may also be used if the connection time-out multiplier produces a time-out value that is not supported by the device or the production inhibit time is not valid.
It is highly recommended to use Extended Status 0x0112 when the RPI values(s) are not acceptable.
Use of this extended status code when the connection time-out multiplier is not supported is deprecated. It is highly recommended that 0x0133 be used instead.
Use of this extended status code when the production inhibit time is not valid is deprecated. It is highly recommended that 0x011B be used instead.</td></tr>
<tr><td rowspan="6">0x01</td><td rowspan="6">0x0112</td><td rowspan="6">Router
Target</td><td colspan="3">RPI VALUE(S) NOT ACCEPTABLE
This extended status code shall be returned when the RPI value(s) in the Forward Open request are outside the range required by the application in the target device or the target is producing at a different interval. The target shall include information with acceptable RPI(s). For this error, the extended status size is 6 16-bit words and is formatted as follows.</td></tr>
<tr><td>Data type</td><td>Value</td><td>Explanation of field</td></tr>
<tr><td>UINT</td><td>0x0112</td><td>Extended status code</td></tr>
<tr><td>USINT</td><td>variable</td><td>Acceptable Originator to Target RPI (see below) type:
0 – the RPI specified in the Forward Open was acceptable (the Originator to Target RPI value is ignored). [a]
1 – unspecified (used to suggest an alternate RPI, e.g. default)
2 – minimum acceptable RPI (used when RPI was too fast for range)
3 – maximum acceptable RPI (used when RPI was too slow for range)
4 – required RPI to correct mismatch (used when data already being consumed at a different interval)
5 to 255 – reserved</td></tr>
<tr><td>USINT</td><td>variable</td><td>Acceptable Target to Originator RPI (see below) type:
0 – the RPI specified in the Forward Open was acceptable (the Target to Originator RPI value is ignored). [a]
1 – unspecified (used to suggest an alternate RPI, e.g.default)
2 – minimum acceptable RPI (used when RPI was too fast for range)
3 – maximum acceptable RPI (used when RPI was too slow for range)
4 – required RPI to correct mismatch (used when data already being produced at a different interval, typically multicast)
5 to 255 – reserved</td></tr>
<tr><td>UDINT</td><td>variable</td><td>Value of Originator to Target RPI that is within the acceptable range for the application. This field is defined the same as the RPI parameter in the Forward Open request.</td></tr>
<tr><td>UDINT</td><td>variable</td><td>Value of Target to Originator RPI that is within the acceptable range for the application. This field is defined the same as the RPI parameter in the Forward Open request.</td></tr>
</table>

General Status	Extended Status	Detected by	Explanation and description
			[a] The value of the Originator to Target and Target to Originator types shall not both be 0.
0x01	0x0113	Originator Router Target	OUT OF CONNECTIONS Connection Manager cannot support any more connections. The maximum number of connections supported by the Connection Manager has already been created.
0x01	0x0114	Router Target	VENDOR ID OR PRODUCT CODE MISMATCH The Product Code or Vendor Id specified in the electronic key logical segment does not match the Product Code or Vendor Id of the device. If the compatibility bit is set this extended status code is returned when the device cannot emulate the specified Vendor ID or Product Code.
0x01	0x0115	Router Target	DEVICE TYPE MISMATCH The Device Type specified in the electronic key logical segment does not match the Device Type of the device. If the compatibility bit is set this extended status code is returned when the device cannot emulate the specified Device Type.
0x01	0x0116	Router Target	REVISION MISMATCH The major and minor revision specified in the electronic key logical segment does not correspond to a valid revision of the device. If the compatibility bit is set this extended status code is returned when the device cannot emulate the specified Major Revision.
0x01	0x0117	Target	INVALID PRODUCED OR CONSUMED APPLICATION PATH The produced or consumed application path specified in the connection path does not correspond to a valid produced or consumed application path within the target application. This error could also be returned if a produced or consumed application path was required, but not provided by a connection request. NOTE This extended status code is "deprecated". It is highly recommended that 0x012A, 0x012B, or 0x012F be used instead.
0x01	0x0118	Target	INVALID OR INCONSISTENT CONFIGURATION APPLICATION PATH An application path specified for the configuration data does not correspond to a configuration application or is inconsistent with the consumed or produced application paths. For example the connection path specifies float configuration data while the produced or consumed paths specify integer data. NOTE This extended status code is "deprecated". It is highly recommended that 0x0129 or 0x012F be used instead.
0x01	0x0119	Target	NON-LISTEN ONLY CONNECTION NOT OPENED Connection request fails since there are no non-listen only connection types currently open. See IEC 61158-5-2, 6.3.1.4.5, for a description of application connection types. The extended status code shall be returned when an attempt is made to establish a listen only connection type to a target, which has no non-listen only connection already established.
0x01	0x011A	Target	TARGET OBJECT OUT OF CONNECTIONS The maximum number of connections supported by this instance of the target object has been exceeded. For example, the Connection Manager could support 20 connections while the target object can only support 10 connections. On the 11th Connection Request to the target object, this extended status code would be used to signify that the maximum number of connections already exist to the target object.
0x01	0x011B	Target	RPI IS SMALLER THAN THE PRODUCTION INHIBIT TIME The Target to Originator RPI is smaller than the Target to Originator Production Inhibit Time.
0x01	0x011C	Router Target	TRANSPORT CLASS NOT SUPPORTED The transport class requested in the Transport Type/Trigger parameter is not supported.

<table>
<tr><th>General Status</th><th>Extended Status</th><th>Detected by</th><th>Explanation and description</th></tr>
<tr><td>0x01</td><td>0x011D</td><td>Router
Target</td><td>PRODUCTION TRIGGER NOT SUPPORTED
The production trigger requested in the Transport Type/Trigger parameter is not supported.</td></tr>
<tr><td>0x01</td><td>0x011E</td><td>Target</td><td>DIRECTION NOT SUPPORTED
The direction requested in the Transport Type/Trigger parameter is not supported.</td></tr>
<tr><td>0x01</td><td>0x011F</td><td>Target</td><td>INVALID ORIGINATOR TO TARGET NETWORK CONNECTION FIXVAR
This extended status code shall be returned as the result of specifying an O⇒T fixed / variable flag that is not supported.</td></tr>
<tr><td>0x01</td><td>0x0120</td><td>Target</td><td>INVALID TARGET TO ORIGINATOR NETWORK CONNECTION FIXVAR
This extended status code shall be returned as the result of specifying a T⇒O fixed / variable flag that is not supported.</td></tr>
<tr><td>0x01</td><td>0x0121</td><td>Target</td><td>INVALID ORIGINATOR TO TARGET NETWORK CONNECTION PRIORITY
This extended status code shall be returned as the result of specifying an O⇒T priority code that is not supported.</td></tr>
<tr><td>0x01</td><td>0x0122</td><td>Target</td><td>INVALID TARGET TO ORIGINATOR NETWORK CONNECTION PRIORITY
This extended status code shall be returned as the result of specifying a T⇒O priority code that is not supported.</td></tr>
<tr><td>0x01</td><td>0x0123</td><td>Target</td><td>INVALID ORIGINATOR TO TARGET NETWORK CONNECTION TYPE
This extended status code shall be returned as the result of specifying an O⇒T connection type that is not supported.</td></tr>
<tr><td>0x01</td><td>0x0124</td><td>Router
Target</td><td>INVALID TARGET TO ORIGINATOR NETWORK CONNECTION TYPE
This extended status code shall be returned as the result of specifying a T⇒O connection type that is not supported.</td></tr>
<tr><td>0x01</td><td>0x0125</td><td>Router
Target</td><td>INVALID ORIGINATOR TO TARGET NETWORK CONNECTION REDUNDANT_OWNER
This extended status code shall be returned as the result of specifying an O⇒T Redundant Owner flag that is not supported.</td></tr>
<tr><td>0x01</td><td>0x0126</td><td>Target</td><td>INVALID CONFIGURATION SIZE
This extended status code is returned when the target device determines that the data segment provided in the Connection_Path parameter did not contain an acceptable number of 16-bit words for the configuration application path requested.
An additional status word shall follow indicating the maximum configuration size supported.
<table>
<tr><th>Data type</th><th>Value</th><th>Explanation of field</th></tr>
<tr><td>UINT</td><td>0x0126</td><td>Extended Status Code</td></tr>
<tr><td>UINT</td><td>Size</td><td>Max size in words</td></tr>
</table></td></tr>
<tr><td>0x01</td><td>0x0127</td><td>Router
Target</td><td>INVALID ORIGINATOR TO TARGET SIZE
This extended status code is returned by the target when the size of the consuming object declared in the Forward_Open request and available on the target does not match the size declared in the O⇒T Network Connection Parameter.
This extended status code is returned by a router when it cannot support the size requested in the O⇒T Network Connection Parameter.
An additional status word shall follow indicating the maximum originator to target size supported.
<table>
<tr><th>Data type</th><th>Value</th><th>Explanation of field</th></tr>
<tr><td>UINT</td><td>0x0127</td><td>Extended Status Code</td></tr>
<tr><td>UINT</td><td>Size</td><td>Max size in octets</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>General Status</th><th>Extended Status</th><th>Detected by</th><th>Explanation and description</th></tr>
<tr><td>0x01</td><td>0x0128</td><td>Router
Target</td><td>INVALID TARGET TO ORIGINATOR SIZE
This extended status code is returned by the target when the size of the producing object declared in the Forward Open request and available on the target does not match the size declared in the T⇒O Network Connection Parameter.
This extended status code is returned by a router when it cannot support the size requested in the T⇒O Network Connection Parameter.
An additional status word shall follow indicating the maximum target to originator size supported.
<table>
<tr><th>Data type</th><th>Value</th><th>Explanation of field</th></tr>
<tr><td>UINT</td><td>0x0128</td><td>Extended Status Code</td></tr>
<tr><td>UINT</td><td>Size</td><td>Max size in octets</td></tr>
</table></td></tr>
<tr><td>0x01</td><td>0x0129</td><td>Target</td><td>INVALID CONFIGURATION APPLICATION PATH
The configuration application path specified in the connection path does not correspond to a valid configuration application path within the target application. This error could also be returned if a configuration application path was required, but not provided by a connection request.</td></tr>
<tr><td>0x01</td><td>0x012A</td><td>Target</td><td>INVALID CONSUMING APPLICATION PATH
The consumed application path specified in the connection path does not correspond to a valid consumed application path within the target application. This error could also be returned if a consumed application path was required, but not provided by a connection request.</td></tr>
<tr><td>0x01</td><td>0x012B</td><td>Target</td><td>INVALID PRODUCING APPLICATION PATH
The produced application path specified in the connection path does not correspond to a valid produced application path within the target application. This error could also be returned if a produced application path was required, but not provided by a connection request.</td></tr>
<tr><td>0x01</td><td>0x012C</td><td>Target</td><td>CONFIGURATION SYMBOL DOES NOT EXIST
Configuration Symbol does not exist. The originator attempts to connect to a configuration tag name, but the name is not on the list of tags defined on the target.</td></tr>
<tr><td>0x01</td><td>0x012D</td><td>Target</td><td>CONSUMING SYMBOL DOES NOT EXIST
Consuming Symbol does not exist. The originator attempts to connect to a consuming tag name, but the name is not on the list of tags defined on the target.</td></tr>
<tr><td>0x01</td><td>0x012E</td><td>Target</td><td>PRODUCING SYMBOL DOES NOT EXIST
Producing Symbol does not exist. The originator attempts to connect to a producing tag name, but the name is not on the list of tags defined on the target.</td></tr>
<tr><td>0x01</td><td>0x012F</td><td>Target</td><td>INCONSISTENT APPLICATION PATH COMBINATION
The combination of configuration and/or consume and/or produce application paths specified in the connection path are inconsistent with each other.</td></tr>
<tr><td>0x01</td><td>0x0130</td><td>Target</td><td>INCONSISTENT CONSUME DATA FORMAT
Information in the data segment is not consistent with the format of the consumed data. For example the configuration data specifies float configuration data while the consumed path specifies integer data.</td></tr>
<tr><td>0x01</td><td>0x0131</td><td>Target</td><td>INCONSISTENT PRODUCE DATA FORMAT
Information in the data segment is not consistent with the format of the produced data. For example the configuration data specifies float configuration data while the produced path specifies integer data.</td></tr>
<tr><td>0x01</td><td>0x0132</td><td>Target</td><td>NULL FORWARD OPEN FUNCTION NOT SUPPORTED
The target does not support the function requested by the Null Forward Open. The requested function may be “ping a device”, “configure a device’s application”, or “reconfigure a target device’s application”.</td></tr>
</table>

General Status	Extended Status	Detected by	Explanation and description
0x01	0x0133	Target Router	CONNECTION TIMEOUT MULTIPLIER NOT ACCEPTABLE This extended status code shall be returned as the result of specifying a connection timeout multiplier value that is reserved or that produces a timeout value that is too large to support in the device.
0x01	0x0134 to 0x0202		Reserved
0x01	0x0203	Originator	CONNECTION TIMED OUT This extended status code shall occur when a connection has timed-out.
0x01	0x0204	Originator Router	UNCONNECTED REQUEST TIMED OUT The Unconnected Request Timed Out error shall occur when the UCMM times out before a response is received. This may occur for an Unconnected_Send, Forward_Open, or Forward_Close service. This typically means that the UCMM has tried a link specific number of times using a link specific retry timer and has not received an acknowledgement or response. This may be the result of congestion at the destination node or may be the result of a node not being powered up or present.
0x01	0x0205	Router	PARAMETER ERROR IN UNCONNECTED REQUEST SERVICE For example, this shall be caused by a Connection Tick Time (see IEC 61158-5-2, 6.2.3.2.1.7) and Connection time-out combination in an Unconnected_Send, Forward_Open, or Forward_Close service that is not supported by an intermediate node.
0x01	0x0206	Originator Router	MESSAGE TOO LARGE FOR UNCONNECTED_SEND SERVICE This shall be caused when the Unconnected_Send is too large to be sent out on a network.
0x01	0x0207	Originator Router	UNCONNECTED ACKNOWLEDGE WITHOUT RESPONSE The message was sent via the unconnected message service and an acknowledge was received but a data response message was not received.
0x01	0x0208 to 0x0300		Reserved
0x01	0x0301	Originator Router Target	NO BUFFER MEMORY AVAILABLE This extended status code shall occur when insufficient connection buffer memory is available in the device.
0x01	0x0302	Originator Router Target	LINK TRANSMIT TIME NOT AVAILABLE FOR DATA This extended status code shall be returned by any device in the path that is a producer and can not allocate sufficient link transmit time for the connection on its link. This can only occur for connections that are specified as scheduled priority.
0x01	0x0303	Originator Router Target	NO CONSUMED CONNECTION ID FILTER AVAILABLE Any device in the path that contains a link consumer for the connection and does not have an available consumed_connection_id filter available shall return this extended status code.
0x01	0x0304	Originator Router Target	NOT CONFIGURED TO SEND SCHEDULED PRIORITY DATA If requested to make a connection that specifies scheduled priority, any device that is unable to send packets during the scheduled portion of the network update time interval shall return this extended status code. For example, on CP 2/1 this code shall be returned by a node whose MAC ID is greater than maximum scheduled node (SMAX).
0x01	0x0305	Router	SCHEDULE SIGNATURE MISMATCH This extended status code shall be returned when the connection scheduling information in the originator device is not consistent with the connection scheduling information on the target network.
0x01	0x0306	Router	SCHEDULE SIGNATURE VALIDATION NOT POSSIBLE This extended status code shall be returned when the connection scheduling information in the originator device can not be validated on the target network. For example, on CP 2/1 this code shall be returned when there is no Keeper in the master state.

General Status	Extended Status	Detected by	Explanation and description
0x01	0x0307 to 0x0310		Reserved
0x01	0x0311	Originator Router	PORT NOT AVAILABLE A Port specified in a Port Segment is Not Available or does not exist.
0x01	0x0312	Originator Router	LINK ADDRESS NOT VALID Link Address specified in Port Segment Not Valid This extended status code is the result of a port segment that specifies a link address that is not valid for the target network type. This extended status code shall not be used for link addresses that are valid for the target network type but do not respond.
0x01	0x0313 to 0x0314		Reserved
0x01	0x0315	Originator Router Target	INVALID SEGMENT IN CONNECTION PATH Invalid Segment Type or Segment Value in Connection Path This extended status code is the result of a device being unable to decode the connection path. For example, this could be caused by an unrecognized path type or a segment type occurring unexpectedly. This extended status code shall only be used when no other more specific extended status code provided in this table applies.
0x01	0x0316	Router Target	FORWARD CLOSE SERVICE CONNECTION PATH MISMATCH The connection path in the Forward_Close service does not match the connection path in the connection being closed. This extended status error code has been "deprecated" because the Forward_Close service uses the connection triad for matching and doesn't use the connection path.
0x01	0x0317	Originator Router Target	SCHEDULING NOT SPECIFIED Either the Schedule Network Segment was not present or the Encoded Value in the Schedule Network Segment is invalid (i.e. 0).
0x01	0x0318	Originator Router	LINK ADDRESS TO SELF INVALID Under some conditions (depends on the device), a link address in the Port Segment which points to the same device (loopback to yourself) is invalid.
0x01	0x0319	Router Target	SECONDARY RESOURCES UNAVAILABLE In a dual chassis redundant system, a connection request that is made to the primary system shall be duplicated on the secondary system. If the secondary system is unable to duplicate the connection request, then this extended status code shall be returned.
0x01	0x031A	Target	RACK CONNECTION ALREADY ESTABLISHED A request for a module connection has been refused because part of the corresponding data is already included in a rack connection.
0x01	0x031B	Target	MODULE CONNECTION ALREADY ESTABLISHED A request for a rack connection has been refused because part of the corresponding data is already included in a module connection.
0x01	0x031C	Originator Router Target	MISCELLANEOUS This extended status is returned when no other extended status code applies for a connection related error.

General Status	Extended Status	Detected by	Explanation and description
0x01	0x031D	Target	REDUNDANT CONNECTION MISMATCH This extended status code shall be returned when the following fields do not match when attempting to establish a redundant owner connection to the same target path: O=>T_RPI; O=>T_connection_parameters; T=>O_RPI; T=>O_connection_parameters; xport_type_and_trigger.
0x01	0x031E	Target	NO MORE USER CONFIGURABLE LINK CONSUMER RESOURCES AVAILABLE IN THE PRODUCING MODULE A target shall return this extended status when the configured number of consumers for a producing application are already in use.
0x01	0x031F	Target	NO USER CONFIGURABLE LINK CONSUMER RESOURCES CONFIGURED IN THE PRODUCING MODULE A target shall return this extended status when there are no consumers configured for a producing application to use.
0x01	0x0320 to 0x07FF		Vendor specific
0x01	0x0800	Originator Router	NETWORK LINK OFFLINE Network link in path to module is offline
0x01	0x0801 to 0x080F		Reserved
0x01	0x0810	Target	NO TARGET APPLICATION DATA AVAILABLE This extended status code is returned when the target application does not have valid data to produce for the requested connection.
0x01	0x0811	Originator	NO ORIGINATOR APPLICATION DATA AVAILABLE This extended status code is returned when the originator application does not have valid data to produce for the requested connection.
0x01	0x0812	Originator Router	NODE ADDRESS HAS CHANGED SINCE THE NETWORK WAS SCHEDULED A router on a scheduled network (e.g. CP 2/1) has a different node address than the value configured in the connection originator.
0x01	0x0813	Router Target	NOT CONFIGURED FOR OFF-SUBNET MULTICAST A multicast connection has been requested between a producer and a consumer that are on different subnets, and the producer is not configured for off-subnet multicast.
0x01	0x0814	Target	INVALID PRODUCE/CONSUME DATA FORMAT Information in the data segment indicates that the format of the produced and/or consumed data is not valid. NOTE This extended status code is "deprecated". It is highly recommended that 0x0130 or 0x0131 be used instead.
0x01	0x0815 to 0xFCFF		Reserved
0x01	0x0FD00 to 0xFFFF		Reserved (not to be used)
0x02	Empty	Originator Router	RESOURCE UNAVAILABLE FOR UNCONNECTED_SEND The device lacks the resources to fully process the Unconnected Send Request.
0x04	Empty	Router	PATH SEGMENT ERROR IN UNCONNECTED SEND Indicates the CPF 2 router experienced a parsing error when extracting the Explicit Messaging Request from the Unconnected Send Request Service Data.

General Status	Extended Status	Detected by	Explanation and description
0x09	Index to Element	Target	ERROR IN DATA SEGMENT. This general status code shall be returned when there is an error in the data segment of a forward open. The Extended Status shall be the index to where the error was encountered in the Data Segment (see 4.1.9.7).
0x0C	Optional	Target	OBJECT STATE ERROR This general status code shall be returned when the state of the target object of the connection prevents the service request from being handled. The Extended Status reports the object's present state. The extended status is optional. For example, a target (application) object of the connection may need to be in an edit mode before attributes can be set. This is different from a service being rejected due to the state of the device.
0x10	Optional	Router Target	DEVICE STATE ERROR This general status code shall be returned when the state of the device prevents the service request from being handled. The Extended Status reports the device's present state. The extended status is optional. For example, a controller may have a key switch which when set to the "hard run" state causes Service Requests to several different objects to fail (i.e. program edits). This general status code would then be returned.
0x13	None	Router Target	NOT ENOUGH DATA The service did not supply enough data to perform the specified operation.
0x15	None	Router Target	TOO MUCH DATA The service supplied more data than was expected.

4.1.11.2 OM errors

4.1.11.2.1 General status format

General status used in the service primitives is specified in Table 180.

Table 180 – General status codes

Status code	Name	Description and meaning of status code
0x00	Success	Service was successfully performed by the object specified
0x01	Connection failure	A connection related service failed along the connection path
0x02	Resource unavailable	Resources needed for the object to perform the requested service were unavailable
0x03	Invalid parameter value	See status code 0x20, which is the preferred value to use for this condition
0x04	Path segment error	The path segment identifier or the segment syntax was not understood by the processing node. Path processing shall stop when a path segment error is encountered
0x05	Path destination unknown	The path is referencing an object class, instance or structure element that is not known or is not contained in the processing node. Path processing shall stop when a path destination unknown error is encountered
0x06	Partial transfer	Only part of the expected data was transferred
0x07	Connection lost	The messaging connection was lost
0x08	Service not supported	The requested service was not implemented or was not defined for this class or object instance
0x09	Invalid attribute value	Invalid attribute data detected

Status code	Name	Description and meaning of status code
0x0A	Attribute list error	An attribute in the Get_Attribute_List or Set_Attribute_List response has a non zero status
0x0B	Already in requested mode/state	The object is already in the mode/state being requested by the service
0x0C	Object state conflict	The object cannot perform the requested service in its current mode/state
0x0D	Object already exists	The requested instance of object to be created already exists
0x0E	Attribute not settable	A request to modify a non-modifiable attribute was received
0x0F	Privilege violation	A permission/privilege check failed
0x10	Device state conflict	The device's current mode/state prohibits the execution of the requested service
0x11	Response data too large	The data to be transmitted in the response buffer is larger than the allocated response buffer
0x12	Fragmentation of a primitive value	The service specified an operation that is going to fragment a primitive data value, i.e. half a REAL data type
0x13	Not enough data	The service did not supply enough data to perform the specified operation
0x14	Attribute not supported	The attribute specified in the request is not supported
0x15	Too much data	The service supplied more data than was expected
0x16	Object does not exist	The object specified does not exist in the device
0x17	Service fragmentation sequence not in progress	The fragmentation sequence for this service is not currently active for this data
0x18	No stored attribute data	The attribute data of this object was not saved prior to the requested service
0x19	Store operation failure	The attribute data of this object was not saved due to a failure during the attempt.
0x1A	Routing failure, request packet too large	The service request packet was too large for transmission on a network in the path to the destination. The routing device was forced to abort the service
0x1B	Routing failure, response packet too largek	The service response packet was too large for transmission on a network in the path from the destination. The routing device was forced to abort the service
0x1C	Missing attribute list entry data	The service did not supply an attribute in a list of attributes that was needed by the service to perform the requested behaviour
0x1D	Invalid attribute value list	The service is returning the list of attributes supplied with status information for those attributes that were invalid
0x1E	Embedded service error	An embedded service resulted in an error
0x1F	Vendor specific error	A vendor specific error has been encountered. The extended status code field of the error response defines the particular error encountered. Use of this general status code should only be performed when none of the status codes presented in this table or within an object class definition accurately reflect the error
0x20	Invalid parameter	A parameter associated with the request was invalid. This code is used when a parameter does not meet the requirements of this specification and/or the requirements defined in an application object specification
0x21	Write once value or medium already written	An attempt was made to write to a write once medium (e.g. WORM drive, PROM) that has already been written, or to modify a value that cannot be changed once established
0x22	Invalid Response Received	An invalid response is received (e.g. response service code does not match the request service code, or response message is shorter than the minimum expected reply size). This status code can serve for other causes of invalid responses
0x23	Buffer overflow	The message received is larger than the receiving buffer can handle. The entire message was discarded.

Status code	Name	Description and meaning of status code
0x24	Message format error	The format of the received message is not supported by the server.
0x25	Key Failure in path	The Key Segment which was included as the first segment in the path does not match the destination module. The extended status shall indicate which part of the key check failed (see Table 181).
0x26	Path Size Invalid	The size of the path which was sent with the Service Request is either not large enough to allow the Request to be routed to an object or too much routing data was included.
0x27	Unexpected attribute in list	An attempt was made to set an attribute that cannot be set at this time.
0x28	Invalid Member ID	The Member ID specified in the request does not exist in the specified Class/Instance/Attribute.
0x29	Member not settable	A request to modify a non-modifiable member was received.
0x2A	Group 2 only server general failure	This status code may only be reported by CP 2/3 Group 2 Only servers with 4 koctets or less code space and only in place of Service not supported, Attribute not supported and Attribute not settable
0x2B	Unknown Type 15 error	A Type 2 to Type 15 translator received an unknown Type 15 Exception Code
0x2C	Attribute not gettable	A request to read a non-readable attribute was received
0x2D	Instance not deletable	The requested object instance cannot be deleted
0x2E to 0xCF	Reserved	Reserved for future extensions
0xD0 to 0xFF	Reserved for object class and service errors	This range of status codes shall be used to indicate object class specific errors. Use of this range should only be performed when none of the status codes presented in this table accurately reflect the error that was encountered

NOTE IEC 61158-5-2 contains more detail on the service response general status codes for each common service.

4.1.11.2.2 Extended status format

The MR_response_Header contains a parameter called Extended_status[] which is labeled extended status data.

Actual usage and definition of this extended status is generally dependant on the object class or service: each object class defines its own extended status values and value ranges (including the vendor specific ones).

Table 181 specifies the format of the extended status for a general status of "Key Failure in path" (0x25).

Table 181 – Extended status code for a general status of "Key Failure in path

General status code	Extended status code	Status name	Explanation and description
0x25	0x0114	VENDOR ID OR PRODUCT CODE MISMATCH	The Product Code or Vendor Id specified in the electronic key logical segment does not match the Product Code or Vendor Id of the target device. If the compatibility bit is set this extended status code is returned when the device cannot emulate the specified Vendor ID or Product Code.
0x25	0x0115	DEVICE TYPE MISMATCH	The Device Type specified in the electronic key logical segment does not match the Device Type of the target device. If the compatibility bit is set this extended status code is returned when the device cannot emulate the specified Device Type.

General status code	Extended status code	Status name	Explanation and description
0x25	0x0116	REVISION MISMATCH	The major and minor revision specified in the electronic key logical segment does not correspond to a valid revision of the target device. If the compatibility bit is set this extended status code is returned when the device cannot emulate the specified Major Revision.
NOTE These extended status codes are the same defined for the Connection Manager general status code 0x01.			

4.1.11.2.3 Object-specific general and extended status format

4.1.11.2.3.1 General

Subclause 4.1.11.2.3 specifies the format of object-specific general and extended status.

4.1.11.2.3.2 Identity object status format

Table 182 specifies the format of object-specific general and extended status for the Identity object.

Table 182 – Identity object status codes

General status code	8-bit associated extended status codes	Status Name	Description of Status
0x00 to 0xCF		General status codes	Defined in 4.1.11.2.1
	0x00 to 0xEE		Reserved extended status codes
	0xF0 to 0xFE	Vendor specific	Vendor specific extended status codes
	0xFF		Used with all general status codes when required and no other extended status code is assigned
0xD0		Hardware diagnostic	Device self testing and hardware diagnostic conditions
	0x00		Reserved
	0x01		Checksum (or CRC) error – Code space/ROM – Boot section
	0x02		Checksum (or CRC) error – Code space/ROM – Application section
	0x03		Checksum (or CRC) error – NV (flash/EEPROM) memory
	0x04		Invalid non-volatile (NV) memory – Configuration bad
	0x05		Invalid non-volatile (NV) memory – No configuration established
	0x06		RAM memory bad – The RAM memory in the device was determined to be experiencing inoperative cells
	0x07		ROM/Flash memory bad
	0x08		Flash/EEPROM (NV) Memory Bad
	0x09		Interconnect wiring error / signal path problem
	0x0A		Power problem – Over current
	0x0B		Power problem – Over voltage
	0x0C		Power problem – Under voltage
	0x0D		Internal sensor problem
	0x0E		System clock fault

General status code	8-bit associated extended status codes	Status Name	Description of Status
	0x0F		Hardware configuration does not match NV configuration
	0x10		Watchdog disabled/idle
	0x11		Watchdog timer expired
	0x12		Device over temperature
	0x13		Ambient temperature outside of operating limits
	0x14 to 0xEF	(reserved)	Reserved
	0xF0 to 0xFE		Vendor specific extended status codes
	0xFF		Used with all general status codes when required and no other extended status code is assigned
0xD1		Device status/states	Device status events and conditions
	0x01		Power applied
	0x02		Device reset
	0x03		Device power loss
	0x04		Activated
	0x05		Deactivated
	0x06		Enter self-test state
	0x07		Enter standby state
	0x08		Enter operational state
	0x09		Non-specific minor recoverable fault detected
	0x0A		Non-specific minor unrecoverable fault detected
	0x0B		Non-specific major recoverable fault detected
	0x0C		Non-specific major unrecoverable fault detected
	0x0D		Fault(s) corrected
	0x0E		Ccv changed
	0x0F		Heartbeat interval changed
	0x10 to 0xEF	(reserved)	
	0xF0 to 0xFE	Vendor specific	Vendor specific
	0xFF		Used with all general status codes when required and no other extended status code is assigned
0xD2 to 0xEF		Object specific general status codes	Reserved – Not yet assigned
	0x00 to 0xFF	Reserved	
0xF0 to 0xFF		Vendor specific general status codes	A vendor specific error has been encountered. The extended status code field of the error response defines the particular error encountered. Use of this general status code should only be performed when none of the status codes presented in this table or within an object class definition accurately reflect the error
	0x00 to 0xFF	Vendor specific extended status codes	All extended status codes are available for association with each vendor specific general status code

4.2 Data abstract syntax specification

4.2.1 Transport format specification

The lower layers of open system architectures are concerned with the transport of user data among distributed functional units. In these layers, the user data may be regarded simply as a sequence of octets. However, application layer entities may manipulate the values of quite complex data types. To achieve independence between the application layer and lower layers, data types may be specified in an abstract syntax notation.

Supplementing the abstract syntax with one or more algorithms (called encoding rules) may determine the values of the lower layer octets which carry the application layer values. The combination of the abstract syntax with a single set of transfer rules produces a specific transfer syntax.

4.2.2 Abstract syntax notation

The data type definitions provided in this STANDARD shall be written in Abstract Syntax Notation One (ASN.1), as defined in ISO/IEC 8824-1. These type definitions shall be a part of the ASN.1 module “Network DataTypes.” The beginning ASN.1 statement indicating that these definitions are in this module is:

```
control network DataTypes DEFINITIONS::= BEGIN
```

and the closing ASN.1 statement shall be the keyword “END”.

The abstract definitions that follow shall comprise the set of control network data types. In addition, provision is made to extend or derive new data types based on existing defined types, and to include those in a “type dictionary.”

4.2.3 Control network data specification

The notation [typeId] for directly derived, enumerated, subrange and structured bit string data shall mean that the tag shall be taken from the “type” field in the corresponding VariableDictionaryEntry.

```
Network Data::= CHOICE{ElementaryData, DerivedData}

ElementaryData::= CHOICE{
            BOOL,
            FixedLengthInteger,
            FixedLengthReal,
            AnyTime,
            AnyDate,
            AnyString,
            FixedLengthBitString,
            EPATH}

DerivedData::= CHOICE {
            DirectlyDerivedData,
            EnumeratedData,
            SubrangeData,
            StructuredBitStringData,
            ARRAY,
            STRUCT,
            FunctionBlockData}

DirectlyDerivedData::= [typeId] NetworkData

EnumeratedData::= [typeId] USINT

SubrangeData::= [typeId] FixedLengthInteger

StructuredBitStringData::= [typeId] FixedLengthBitString
```

```
FixedLengthInteger::= CHOICE {SignedInteger, UnsignedInteger}
SignedInteger::= CHOICE {SINT, INT, DINT, LINT}
UnsignedInteger::= CHOICE {USINT, UINT, UDINT, ULINT}
FixedLengthReal::= CHOICE {REAL, LREAL}
AnyTime::= CHOICE {ITIME,TIME, FTIME, LTIME}
AnyDate::= CHOICE {DATE, TIME_OF_DAY, DATE_AND_TIME}
AnyString::= CHOICE {STRING, STRING2}
FixedLengthBitString::= CHOICE {SWORD, WORD, DWORD, LWORD}
BOOL::= [PRIVATE 1] IMPLICIT BOOLEAN
SINT::= [PRIVATE 2] IMPLICIT OCTET STRING-- 1 octet
INT::= [PRIVATE 3] IMPLICIT OCTET STRING-- 2 octets
DINT::= [PRIVATE 4] IMPLICIT OCTET STRING-- 4 octets
LINT::= [PRIVATE 5] IMPLICIT OCTET STRING-- 8 octets
USINT::= [PRIVATE 6] IMPLICIT OCTET STRING-- 1 octet
UINT::= [PRIVATE 7] IMPLICIT OCTET STRING-- 2 octets
UDINT::= [PRIVATE 8] IMPLICIT OCTET STRING-- 4 octets
ULINT::= [PRIVATE 9] IMPLICIT OCTET STRING-- 8 octets
REAL::= [PRIVATE 10] IMPLICIT OCTET STRING-- 4 octets
LREAL::= [PRIVATE 11] IMPLICIT OCTET STRING-- 8 octets
STIME::= [PRIVATE 12] IMPLICIT DINT
DATE::= [PRIVATE 13] IMPLICIT UINT
TIME_OF_DAY::= [PRIVATE 14] IMPLICIT UDINT
DATE_AND_TIME::= [PRIVATE 15] IMPLICIT SEQUENCE {
 time_of_day  UDINT,
 date  UINT }
STRING::= [PRIVATE 16] IMPLICIT SEQUENCE {
            charcount      UINT,
            stringcontents OCTET STRING} -- one octet per character
SWORD::= [PRIVATE 17] IMPLICIT OCTET STRING-- 1 octet
WORD::= [PRIVATE 18] IMPLICIT OCTET STRING-- 2 octets
DWORD::= [PRIVATE 19] IMPLICIT OCTET STRING-- 4 octets
LWORD::= [PRIVATE 20] IMPLICIT OCTET STRING-- 8 octets
STRING2::= [PRIVATE 21] IMPLICIT SEQUENCE {
            charcount      UINT,
            string2contents OCTET STRING} -- 2 octets/ character
FTIME::= [PRIVATE 22] IMPLICIT DINT
LTIME::= [PRIVATE 23] IMPLICIT LINT
ITIME::= [PRIVATE 24] IMPLICIT INT
STRINGN::= [PRIVATE 25] IMPLICIT SEQUENCE {
            charsize    UINT,
            charcount   UINT,
            stringNcontents OCTET STRING} -- N octets/ character
```

```
SHORT_STRING::= [PRIVATE 26] IMPLICIT SEQUENCE {
            charcount      USINT,
            stringcontents OCTET STRING} -- one octet per character

TIME        ::=   [PRIVATE 27] IMPLICIT DINT

EPATH       ::=   [PRIVATE 28] IMPLICIT OCTET STRING  -- IEC 61158-6-2

STRINGI     ::=   [PRIVATE 30] IMPLICIT SEQUENCE{
            Stringnum   USINT (number of strings)
            array of    STRUCT
            language1   USINT (first character from ISO 639-2/T)
            language2   USINT (second character from ISO 639-2/T)
            language3   USINT (third character from ISO 639-2/T)
            datatype    EPATH limited to the values 0xD0,0xD5,0xD9,and
0xDA)
            charset     UINT     from IANA MIB Printer Codes
(IETF RFC 1759))
            stringcontents    CHOICE OF (SHORT_STRING, STRING, STRING2,
                        or STRINGN) -- based on datatype field

ARRAY::= SEQUENCE OF NetworkData -- All of same base type

STRUCT::= SEQUENCE OF NetworkData -- May be different types

FunctionBlockData::= SET{
            inputs  [0] IMPLICIT STRUCT OPTIONAL,
            outputs     [1] IMPLICIT STRUCT OPTIONAL,
            controlInputs  [2] IMPLICIT STRUCT OPTIONAL,
            controlOutputs [3] IMPLICIT STRUCT OPTIONAL}
```

4.2.4 Data type specification / dictionaries

The definition of an object may include text that defines attributes. Attributes shall be assigned a **Data Type** in an object specification. The Data Type may be one of those defined in this standard or may be an object specific extension to this standard. The following definition shall provide a **Type Specification** for data and shall provide a structure for extending or deriving new data types based on existing defined types.

```
Dictionary::= CHOICE {VariableDictionary, TypeDictionary}

VariableDictionary::= SEQUENCE OF VariableDictionaryEntry

VariableDictionaryEntry::= SEQUENCE{
            name  AnyString,
            id FixedLengthInteger,
            type  TypeID,
            ranges   SEQUENCE OF Subrange,-- for arrays
            accessPrivilege   BOOL {READ_ONLY(0), READ_WRITE(1)}
TypeID::= OCTET STRING  -- ASN.1 encoded tag value of the

-- DataTypeSpecification module

Subrange::= SEQUENCE {
            minValue FixedLengthInteger,
            maxValue FixedLengthInteger}

TypeDictionary::= SEQUENCE OF TypeDictionaryEntry

TypeDictionaryEntry::= SEQUENCE {
            name  AnyString,
            type  TypeID,
            spec  DataTypeSpecification}
```

```
DataTypeSpecification::= CHOICE {
            alt      [PRIVATE 0] IMPLICIT AlternateTypeSpec,
            bool  [PRIVATE 1] IMPLICIT NULL, -- BOOL
            sint  [PRIVATE 2] IMPLICIT NULL, -- SINT
            int   [PRIVATE 3] IMPLICIT NULL, -- INT
            dint  [PRIVATE 4] IMPLICIT NULL, -- DINT
            lint  [PRIVATE 5] IMPLICIT NULL, --    LINT
            usint [PRIVATE 6] IMPLICIT NULL, --    USINT
            uint  [PRIVATE 7] IMPLICIT NULL, --    UINT
            udint [PRIVATE 8] IMPLICIT NULL, --    UDINT
            ulint [PRIVATE 9] IMPLICIT NULL, --    ULINT
            real  [PRIVATE 10]   IMPLICIT NULL, --    REAL
            lreal [PRIVATE 11]   IMPLICIT NULL, --    LREAL
            stime [PRIVATE 12]   IMPLICIT NULL, --    STIME
            date  [PRIVATE 13]   IMPLICIT NULL, --    DATE
            tod   [PRIVATE 14]   IMPLICIT NULL, --    TIME_OF_DAY
            dat   [PRIVATE 15]   IMPLICIT NULL, --    DATE_AND_TIME
            str1  [PRIVATE 16]   IMPLICIT NULL, --    STRING
            sword [PRIVATE 17]   IMPLICIT NULL, --    SWORD
            word  [PRIVATE 18]   IMPLICIT NULL, --    WORD
            dword [PRIVATE 19]   IMPLICIT NULL, --    DWORD
            lword [PRIVATE 20]   IMPLICIT NULL, --    LWORD
            str2  [PRIVATE 21]   IMPLICIT NULL, --    STRING2
            ftime [PRIVATE 22]   IMPLICIT NULL, --    FTIME
            ltime [PRIVATE 23]   IMPLICIT NULL, --    LTIME
            itime [PRIVATE 24]   IMPLICIT NULL, --    ITIME
            strN  [PRIVATE 25]   IMPLICIT NULL, --    STRINGN
            shstr [PRIVATE 26]   IMPLICIT NULL, --    SHORT_STRING
            time     [PRIVATE 27]   IMPLICIT NULL, --    TIME
            epath    [PRIVATE 28]   IMPLICIT NULL, --    EPATH
            strI     [PRIVATE 30]   IMPLICIT NULL, -- STRINGI
            constructedData   CHOICE {
               abbrvStruc  [0]   IMPLICIT AbbreviatedStrucTypeSpec,
               abbrvArr    [1]   IMPLICIT AbbreviatedArrayTypeSpec,
               frmlStruc   [2]   IMPLICIT FormalStrucTypeSpec,
               frmlArr     [3]   IMPLICIT FormalArrayTypeSpec,
               expBitStr   [4]   IMPLICIT ExpandedFixedLenBitStrTypeSpec,
               expStr1     [5]   IMPLICIT ExpandedStringTypeSpec,
               expStr2     [6]   IMPLICIT ExpandedString2TypeSpec
               frm1HandleStruc [7]  IMPLICIT  FormalHandleStrucTypeSpec}
}

AbbreviatedStrucTypeSpec::= UINT

AbbreviatedArrayTypeSpec::= DataTypeSpecification

FormalStrucTypeSpec::= SEQUENCE OF DataTypeSpecification

FormalHandleStrucTypeSpec::= SEQUENCE OF DataTypeHandleSpecification

DataTypeHandleSpecification::= DataTypeSpecification MemberHandle

MemberHandle::= USINT, UINT, UDINT
FormalArrayTypeSpec::= SEQUENCE {
            lowBound [0] IMPLICIT FixedLengthInteger, -- Array Lower Bound
            highBound   [1] IMPLICIT FixedLengthInteger, -- Array Upper
Bound
            dataType    DataTypeSpecification }
```

```
ExpandedFixedLenBitStrTypeSpec::= SEQUENCE {
            bitStrType DataTypeSpecification    -- SWORD, WORD, DWORD, or
LWORD
            bitFields [7] IMPLICIT BitFieldDef}

BitFieldDef::= SEQUENCE OF {
            bitDef [2] IMPLICIT OCTET STRING}   -- Length is always 2
octets.
                                        -- First octet contains starting
                                        -- Bit Position. Trailing octet
                                        -- contains the number of bits.

ExpandedStringTypeSpec::= UINT-- String Length In Octets

ExpandedString2TypeSpec::= UINT-- String Length In Octets
AlternateTypeSpec::= CHOICE {
            directlyDerivedTypeSpec [0]   IMPLICIT    TypeID,
            subrangeTypeSpec        [1]   IMPLICIT    SubrangeTypeSpec ,
            enumeratedTypeSpec      [2]   IMPLICIT    EnumeratedTypeSpec,
            fbTypeSpec              [3]   IMPLICIT    FBTypeSpec}

SubrangeTypeSpec::= SEQUENCE{
            baseType    TypeID,     -- NOTE  minValue and maxValue
            minValue    FixedLengthInteger,  -- shall be within the range
            maxValue    FixedLengthInteger}  -- of baseType values

EnumeratedTypeSpec::= SEQUENCE OF AnyString

BitNameDefintion::= SEQUENCE {
            bitName  AnyString,
            bitNumber   USINT}
FBTypeSpec::= SET{
            inputs         [0] IMPLICIT FbtElementSpec OPTIONAL,
            outputs        [1] IMPLICIT FbtElementTypeSpec OPTIONAL,
            controlInputs  [2] IMPLICIT FbtElementTypeSpec OPTIONAL,
            controlOutputs [3] IMPLICIT FbtElementTypeSpec OPTIONAL}

FbtElementTypeSpec::= SEQUENCE OF ElementSpec

ElementSpec::= SEQUENCE {
            name     AnyString,
            typespec    ElementTypeSpec}

ElementTypeSpec::= CHOICE {
            [0] IMPLICIT TypeID,
            [1] IMPLICIT SubrangeTypeSpec,
            [2] IMPLICIT EnumeratedTypeSpec,
            [3] IMPLICIT FormalArrayTypeSpec,
            [4] IMPLICIT ExpandedStringTypeSpec,
            [5] IMPLICIT ExpandedString2TypeSpec}
```

The following END statement shall terminate the ASN.1 module opened in 4.1.

END.

4.3 Encapsulation abstract syntax

4.3.1 Encapsulation protocol

4.3.1.1 General

In order to send TPDUs over TCP/IP, an encapsulation protocol is required. Subclause 4.3.1 defines the encapsulation protocol requirements. The encapsulation protocol is a generic protocol that may be used for transporting data other than Type 2 TPDUs.

The encapsulation protocol defines a reserved TCP port number that shall be supported by all Type 2 devices (0xAF12). All Type 2 devices shall accept at least two TCP connections on TCP port number 0xAF12. Once the TCP connection to TCP port number 0xAF12 is established, all data sent through the TCP stream shall be in the format specified in 4.3.1.2 and 4.3.1.3.

NOTE TCP is a stream-based protocol. It can send almost any length IP packet. For example, if two back-to-back encapsulated messages are passed to a TCP/IP stack, the TCP/IP stack can put both encapsulated messages in a single Ethernet frame, or place half of the first message in the first Ethernet frame and all the rest in the next Ethernet frame.

The encapsulation protocol also defines a reserved UDP port number that shall be supported by all EtherNet/IP devices (0xAF12). All devices shall accept UDP packets on UDP port number 0xAF12. Whenever UDP is used to send an encapsulated message, the entire message shall be sent in a single UDP packet. Only one encapsulated message shall be present in a single UDP packet destined to UDP port 0xAF12.

Some encapsulated messages shall only be sent via TCP. Other may be sent via either UDP or TCP. See Table 184 for details about which commands are restricted to TCP.

4.3.1.2 Encapsulation messages

All encapsulation messages sent via TCP or sent via UDP port 0xAF12 shall be composed of a fixed-length header of 24 octets followed by an optional data portion. The total encapsulation message length (including header) shall be limited to 65 535 octets. The encapsulation message length shall not override length restrictions imposed by the encapsulated protocol. Its structure shall be as shown in Figure 15, where the top of the diagram indicates the portion of the message sent first on the wire.

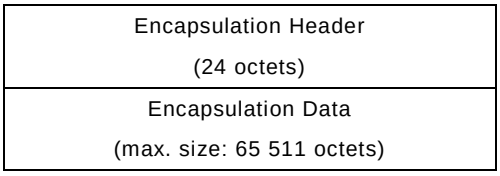


Figure 15 – Encapsulation message

The encapsulation data portion of the message is required only for certain commands.

4.3.1.3 Encapsulation header

The encapsulation header shall be as shown in Table 183.

Table 183 – Encapsulation header

Field name	Format	Description
Command	UINT	Encapsulation command
Length	UINT	Length, in octets, of the command specific data portion of the message, i.e., the number of octets following the header
Session handle	UDINT	Session identification (application dependent)
Status	UDINT	Status code
Sender context	ARRAY of 8 USHORT	Information pertinent only to the sender of an encapsulation command
Options	UDINT	Options flags

Multi-octet integer fields in encapsulation messages shall be transmitted as specified in 4.3.3.4 (i.e. using little endian octet ordering).

NOTE 1 This is different from the octet ordering used in standard Internet network protocols, which is big endian.

Although the header contains no explicit information to distinguish between a request and a reply, this information shall be determined in either of two ways:

- implicitly, by the command and the context in which the message is generated. (For example, in the case of the RegisterSession command, the request is generated by an originator and the target generates the reply) ;
- explicitly, by the contents of an encapsulated protocol packet in the data part of the message.

NOTE 2 The establishment of a session is defined in 11.8. A session makes a TCP/IP connection between originator and target over which encapsulated commands can be sent. Since TCP/IP connections are modeled as a stream of octets, the encapsulation header is added at the beginning of each encapsulated packet so that the receiving device can know where packets begin and end.

4.3.1.4 Command field

The allocation of command codes shall be as shown in Table 184.

Table 184 – Encapsulation command codes

Command code	Name	Usage
0x0000	NOP	May be sent only using TCP
0x0001 to 0x0003	Reserved for legacy usage[a]	
0x0004	ListServices	May be sent using either UDP or TCP
0x0005	Reserved for legacy usage[a]	
0x0006 to 0x0062	Reserved for future extensions[b]	
0x0063	ListIdentity	May be sent using either UDP or TCP
0x0064	ListInterfaces	Optional (may be sent using either UDP or TCP)
0x0065	RegisterSession	May be sent only using TCP
0x0066	UnRegisterSession	May be sent only using TCP
0x0067 to 0x006E	Reserved for legacy usage[a]	
0x006F	SendRRData	May be sent only using TCP
0x0070	SendUnitData	May be sent only using TCP
0x0071	Reserved for legacy usage[a]	
0x0072	IndicateStatus	Optional (may be sent only using TCP)
0x0073	Cancel	Optional (may be sent only using TCP)

Command code	Name	Usage
0x0074 to 0x00C7	Reserved for legacy usage[a]	
0x00C8 to 0xFFFF	Reserved for future extensions[b]	

[a] Commands marked as “Reserved for legacy usage” indicate commands that were defined prior to the publication of this standard. Their behavior is undefined in this standard. Devices shall not implement these commands without prior knowledge of the legacy usage. Devices that do not support these commands shall return encapsulation status code 0x0001.

[b] Commands marked as “Reserved for future extensions” shall not be used.

A device shall accept commands that it does not support without breaking the session or underlying TCP connection. A status code indicating that an unsupported command was received shall be returned to the sender of the message (see 4.3.1.7).

4.3.1.5 Length field

The length field in the header shall specify the size in octets of the data portion of the message. The field shall contain zero for messages that contain no data. The total length of a message shall be the sum of the number contained in the length field plus the 24-octet size of the encapsulation header.

The entire encapsulation message shall be read from the TCP/IP connection even if the length is invalid for a particular command or exceeds the host’s internal buffers. Data that would exceed internal buffers may be discarded, however the entire encapsulation messages shall be read.

NOTE Failure to read the entire message can result in losing track of the message boundaries in the TCP octet stream.

4.3.1.6 Session handle field

The Session Handle shall be generated by the target and returned to the originator in response to a RegisterSession request. The originator shall insert it in all subsequent encapsulation command requests (sent using the commands listed in Table 184) which require sessions to that particular target. In the case where the target initiates and sends a command to the originator, the target shall include this field in the request that it sends to the originator.

NOTE Some commands (e.g. Nop) do not require a session handle, even if a session has been established. The specification of a particular command will indicate if it does not require a session.

4.3.1.7 Status field

The value in the Status field shall indicate whether or not the receiver was able to execute the requested encapsulation command. A value of zero in a reply shall indicate successful execution of the command. In all requests issued by the sender, the Status field shall contain zero. If the receiver receives a request with a non-zero Status field, the request shall be ignored and no reply shall be generated.

NOTE This field does not reflect errors that are generated by an encapsulated protocol packet contained within the data portion of the message. For example, an error encountered during an end node’s processing of a Set Attributes service would be returned via the Type 2 specified error mechanism.

The status codes shall be as shown in Table 185.

Table 185 – Encapsulation status codes

Status code	Description
0x00	Success
0x01	The sender issued an invalid or unsupported encapsulation command.
0x02	Insufficient memory resources in the receiver to handle the command. This is not an application error. nstead, it only occurs if the encapsulation layer cannot obtain memory resources that it needs.
0x03	Poorly formed or incorrect data in the data portion of the encapsulation message.
0x04 to 0x63	Reserved for legacy usage[a]
0x64	An originator used an invalid session handle when sending an encapsulation message to the target.
0x65	The target received a message of invalid length (see 4.3.1.5).
0x66 to 0x68	Reserved for legacy usage[a]
0x69	Unsupported protocol version.
0x6A to 0xFFFF	Reserved for future extensions[b]

[a] Status codes marked as “Reserved for legacy usage” indicate status codes that were defined prior to the publication of this standard. Their usage is undefined in this standard. Devices shall not use these status codes without prior knowledge of the legacy usage.

[b] Status codes marked as “Reserved for future extensions” shall not be used.

4.3.1.8 Sender context field

The sender of the command shall assign the value in the Sender Context field of the header. The receiver shall return this value without modification in its reply. Commands with no expected reply may ignore this field.

The sender of a command request may place any value in this field.

NOTE This field can be used to match requests with their associated replies.

4.3.1.9 Options field

The intent of this field is to provide the bits that modify the meaning of the various encapsulation commands. Options and option behavior are defined on a per-command basis.

4.3.1.10 Command specific data field

The structure of the command specific data field depends on the command code. To organize their command specific data field, most commands use either or both of the following two methods:

- use a fixed structure;
- use the common packet format (specified in 4.3.3).

The common packet format allows commands to structure their command specific data field in an extensible way.

4.3.2 Command descriptions

4.3.2.1 Nop

Either an originator or a target may send a Nop request. No reply shall be generated. The data portion of the request shall be from 0 to 65 511 octets long. The receiver shall ignore any

data that is contained in the message. A Nop request does not require that a session be established.

NOTE A Nop provides a way for either an originator or target to determine if the TCP connection is still open.

The Nop request encapsulation header shall be as shown in Table 186.

Table 186 – Nop request encapsulation header

Field name	Data type	Field value
Command	UINT	Nop (0x00)
Length	UINT	Length of the command specific data
Session handle	UDINT	Any value (ignored by target)
Status	UDINT	0 [a]
Sender context	ARRAY of 8 USHORT	Chosen by sender
Options	UDINT	0

[a] For backwards compatibility, no options shall be defined for this command. The receiver shall discard packets with a non-zero option field.

4.3.2.2 RegisterSession

An originator shall send a RegisterSession request to a target to initiate a session. The RegisterSession command does not require that a session be established.

NOTE See 11.8 for detailed information on establishing and maintaining a session.

The RegisterSession request encapsulation header shall be as shown in Table 187.

Table 187 – RegisterSession request encapsulation header

Field name	Data type	Field value
Command	UINT	RegisterSession (0x65)
Length	UINT	4
Session handle	UDINT	Any value (ignored by target)
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Any value
Options	UDINT	0 [a]

[a] For backwards compatibility, no options shall be defined for this command. The receiver shall discard packets with a non-zero option field.

The parameters in the data portion shall determine the version of the protocol and any session options as shown in Table 188. The protocol version shall be set to 1. No option flags are currently defined, so the session options flags shall be set to 0.

NOTE This options flags field is not the same as the options flags in the encapsulation header.

Table 188 – RegisterSession request data portion

Field	Type	Description
Protocol version	UINT	Requested protocol version (1)
Options flags	UINT	Session options (0)

The target shall send a RegisterSession reply to indicate that it has registered the originator. The reply shall have the same format as the request as shown in Table 189.

Table 189 – RegisterSession reply encapsulation header

Field name	Data type	Field value
Command	UINT	RegisterSession (0x65)
Length	UINT	4
Session handle	UDINT	Session handle generated by target
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Value from request
Options	UDINT	0 [a]

[a] For backwards compatibility, no options shall be defined for this command. The receiver shall discard packets with a non-zero option field.

The Session Handle field of the header shall contain a target-generated identifier that the originator shall save and insert in the Session Handle field of the header for all subsequent requests to that target. This field shall be valid only if the Status field is zero (0).

The Sender Context field of the header shall contain the same values present in the original sender request.

If the originator was successfully registered with the target, the Status field shall be zero (0). If the originator was not successfully registered, the Status field shall contain the appropriate status code, as follows:

- status code 0x0001 shall be returned if the originator attempts to register more than 1 active session on the same TCP connection;
- status code 0x0002 shall be returned if the target does not have sufficient resources to register the originator;
- other status codes from Table 185 may be used as appropriate for general encapsulation related errors (e.g., poorly formed encapsulation message, invalid length);
- status code 0x0069 shall be returned for Protocol Version or Options mismatches, as described below.

The data portion of the reply shall have the same format as the request as shown in Table 190.

Table 190 – RegisterSession reply data portion

Field	Type	Description
Protocol Version	UINT	Version from RegisterSession request if supported. If the requested version is not supported, contains the highest version supported.
Options	UINT	Options flags from RegisterSession request if supported. If any requested Options flags are not supported, contains the supported Options flags.

The Protocol Version field shall equal the requested version if the originator was successfully registered. If the target does not support the requested version of the protocol:

- the session shall not be created;
- the Status field shall be set to "unsupported encapsulation protocol" (0x0069);

- the target shall return the highest supported version in the Protocol Version field.

At present, no Options flags are defined. In order to support their future definition, targets shall check the value of the Options flags in the RegisterSession request. If all requested options are supported, the Options field in the reply shall contain the originator's requested value. If the target does not support the requested options:

- the session shall not be created;
- the Status field shall be set to "unsupported encapsulation protocol" (0x0069);
- the target shall return the options that it supports in the RegisterSession reply.

4.3.2.3 UnRegisterSession

Either an originator or a target may send this request to terminate the session. The receiver shall initiate a close of the underlying TCP/IP connection when it receives this request. The session shall also be terminated when the transport connection between the originator and target is terminated. The receiver shall perform any other associated cleanup required on its end. There shall be no reply to this command, except in the event that the command is received via UDP. If the command is received via UDP, the receiver shall reply with encapsulation status code 0x0001 (invalid or unsupported command).

The UnregisterSession request format shall be as shown in Table 191.

Table 191 – UnRegisterSession request encapsulation header

Field name	Data type	Field value
Command	UINT	UnRegisterSession (0x65)
Length	UINT	4
Session Handle	UDINT	Session handle from RegisterSession
Status	UDINT	0
Sender Context	ARRAY of 8 USHORT	Any value (ignored by target)
Options	UDINT	0 [a]

[a] For backwards compatibility, no options shall be defined for this command. The receiver shall discard packets with a non-zero option field.

The receiver shall not reject the UnRegisterSession due to unexpected values in the encapsulation header (invalid Session Handle, non-zero Status, non-zero Options, or additional command data). In all cases the TCP connection shall be closed.

NOTE See 11.8.3 for more details about terminating a session.

4.3.2.4 ListServices

The ListServices request shall be sent to determine which encapsulation service classes the target device supports. The ListServices command does not require that a session be established.

NOTE Each service class has a unique type code, and an optional ASCII name.

The ListServices request encapsulation header shall be as shown in Table 192.

Table 192 – ListServices request encapsulation header

Field name	Data type	Field value
Command	UINT	ListServices (0x04)
Length	UINT	0

Field name	Data type	Field value
Session handle	UDINT	Any value (ignored by target)
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Chosen by sender
Options	UDINT	0 [a]

[a] For backwards compatibility, no options shall be defined for this command. The receiver shall discard packets with a non-zero option field.

The receiver shall reply with an encapsulation message consisting of the header and data, as shown in Table 193 and Table 194. The data portion of the reply shall provide the information on the services supported.

Table 193 – ListServices reply encapsulation header

Field name	Data type	Field value
Command	UINT	ListServices (0x04)
Length	UINT	Length of the command specific data
Session handle	UDINT	Any value (ignored by receiver)
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Value from request
Options	UDINT	0 [a]

[a] For backwards compatibility, no options shall be defined for this command. The receiver shall discard packets with a non-zero option field.

The data portion of the reply shall contain a 2-octet item count followed by an array of items describing the service(s) provided, as shown in Table 194.

Table 194 – ListServices reply data portion

Field name	Type	Description
Item count	UINT	Number of items to follow
Target items	STRUCT of	Interface information
	UINT	Item ID
	UINT	Item length
	UINT	Version of encapsulated protocol (shall be set to 1)
	UINT	Capability Flags
	ARRAY of 16 USINT	Name of service

The Item ID shall identify the service class. One service class is defined, with type code 0x100 and name “Communications”. This service class shall indicate that the device supports encapsulation of Type 2 PDU’s. All devices that support Type 2 PDU encapsulation shall support the ListServices command and Communications service class.

The Version field shall indicate the version of the service supported by the target to help maintain compatibility between applications.

Each service shall have a different set of capability flags. Reserved flags shall be set to zero.

The Capability Flags, defined for the Communications service, shall be as shown in Table 195.

Table 195 – Communications capability flags

Flag value	Description
Bits 0 to 4	Reserved for legacy usage[a]
Bit 5	If the device supports Type 2 PDU encapsulation this bit shall be set (= 1) ; otherwise, it shall be clear (= 0)
Bits 6 to 7	Reserved for legacy usage [a]
Bit 8	Supports transport class 0 or 1 UDP-based connections
Bits 9 to 15	Reserved for future extensions

[a] Flag marked as “Reserved for legacy usage” indicate flags that were defined prior to the publication of this standard. Their usage is undefined in this standard. Devices shall not use these flags without prior knowledge of the legacy usage. If a device receives a reserved flag that it does not understand, the reply shall be processed and the flag ignored.

The Name field shall allow up to a 16-octet, NULL-terminated ASCII string for descriptive purposes only. The 16-octet limit shall include the NULL character.

4.3.2.5 ListIdentity

A connection originator may use the ListIdentity request to locate and identify potential targets. This request shall be sent as a unicast message using TCP or UDP, or as a broadcast message using UDP and does not require that a session be established. The reply shall always be sent as a unicast message.

When received as a broadcast message, the receiving device shall delay for a pseudo-random period of time prior to sending the reply as specified below. Delaying before sending the reply helps to spread out any resulting ARP requests and ListIdentity replies from target devices on the network.

The ListIdentity request encapsulation header shall be as shown in Table 196.

Table 196 – ListIdentity request encapsulation header

Field name	Data type	Field value
Command	UINT	ListIdentity (0x63)
Length	UINT	0
Session handle	UDINT	Any value (ignored by target)
Status	UDINT	0
Sender context	UINT	MaxResponseDelay in ms, see below
	ARRAY of 6 USHORT	Reserved, shall be ignored by the receiver, values shall be 0
Options	UDINT	0 [a]

[a] For backwards compatibility, no options shall be defined for this command. The receiver shall discard packets with a non-zero option field.

One reply item is defined for this command, Target Identity, with item type code 0x0C. This item shall be supported (returned) by all Type 2 devices.

A receiver of the ListIdentity command shall reply with an encapsulation message consisting of the header and data, as shown in Table 197 andTable 198. The data portion of the message shall provide the information on the target’s identity. The reply shall be sent to the IP address from which the request was received.

When the ListIdentity request has been received as a UDP broadcast message, the receiver shall delay before sending the reply. When received as a unicast message (either via UDP or TCP), the receiver shall not delay.

The receiver's delay shall be a random value, in milliseconds, between 0 and the MaxResponseDelay specified in the ListIdentity request. If the sender specifies a MaxResponseDelay value of 0 ms, a default value of 2 000 ms shall be used by the receiver. If the sender specifies a MaxResponseDelay value of 1 ms to 500 ms, a value of 500 ms shall be used by the receiver. A new random value shall be chosen for each request.

The purpose of the delay is to spread the ListIdentity responses (and the ARP messages that may result) over the MaxResponseDelay interval. It is therefore important that each device generate a unique random delay value. Devices shall ensure they each generate a unique value, for example by seeding their random number generators with a unique value such as their IP address or Ethernet MAC address.

It is possible that devices may receive additional broadcast ListIdentity requests while delaying for the response, for example, if multiple clients issue the broadcast ListIdentity request. Devices should be able to accept and process at least 2 outstanding broadcast ListIdentity requests concurrently.

After issuing a broadcast ListIdentity request, a client should not issue further requests until the MaxResponseDelay interval for its current request has expired. Client behavior for handling ListIdentity responses received beyond MaxResponseDelay is vendor specific.

Table 197 – ListIdentity reply encapsulation header

Field name	Data type	Field value
Command	UINT	List Identity (0x63)
Length	UINT	Length of the command specific data
Session handle	UDINT	Any value (ignored by receiver)
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Value from request
Options	UDINT	0 [a]

[a] For backwards compatibility, no options shall be defined for this command. The receiver shall discard packets with a non-zero option field.

The data portion of the reply is structured as a Common Packet Format that contains a 2-octet item count followed by an array of items providing the target identity, as shown in Table 198.

Table 198 – ListIdentity reply data portion (successful)

Field name	Data type	Field value
Item count	UINT	Number of target items to follow
Target items	STRUCT of	Interface information
	UINT	Item ID
	UINT	Item length
	ARRAY of octets	Item data

The CFP 2 Identity item shall be the first item returned and has the format as defined in Table 199. Part of this item definition follows the Get Attribute All service response definition

of the Identity object (data returned based on instance one of this object). Unlike most fields in the Common Packet Format, the Socket Address field shall be sent in big endian order.

At present no additional items are defined for the ListIdentity reply. Additional items may be defined in the future. Receivers of the ListIdentity reply shall ignore unexpected items.

Table 199 – CPF 2 identity item

Field name	Type	Description
Item ID	UINT	Item ID of CPF 2 Identity (0x0C)
Item length	UINT	Number of octets in item which follows (length varies depending on Product Name string)
Version	UINT	Encapsulation protocol version supported (also returned with Register Session reply)
Socket Address	STRUCT of:	Socket Address (see 4.3.3.3.3)
	INT	sin_family (big-endian)
	UINT	sin_port (big-endian)
	UDINT	sin_addr (big-endian)
	ARRAY of USINT	sin_zero (length of 8) (big-endian)
Vendor ID[a]	UINT	Device manufacturers Vendor ID
Device Type[a]	UINT	Device Type of product
Product Code[a]	UINT	Product Code assigned with respect to device type
Revision[a]	USINT[2]	Device revision
Status[a]	WORD	Current status of device
Serial Number[a]	UDINT	Serial number of device
Product Name[a]	SHORT_STRING	Human readable description of device
State[b]	USINT	Current state of device

[a] These parameters are further defined by the corresponding instance attribute of the Identity object.

[b] The State attribute is an optional attribute of the Identity Object. If not implemented, the value shall be 0xFF.

4.3.2.6 ListInterfaces

The optional ListInterfaces request shall be used by a connection originator to identify non-Type 2 communication interfaces associated with the target. A session need not be established to send this command.

The ListInterfaces request encapsulation header shall be as shown in Table 200.

Table 200 – ListInterfaces request encapsulation header

Field name	Data type	Field value
Command	UINT	List Interfaces (0x64)
Length	UINT	0
Session handle	UDINT	Any value (ignored by target)
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Chosen by sender
Options	UDINT	0 [a]

[a] For backwards compatibility, no options shall be defined for this command. The receiver shall discard packets with a non-zero option field.

If supported, the receiver of a ListInterfaces request command shall reply with an encapsulation message consisting of the header and data, as shown below in Table 201.

Table 201 – ListInterfaces reply encapsulation header

Field name	Data type	Field value
Command	UINT	List Interfaces (0x64)
Length	UINT	Length of the command specific data
Session handle	UDINT	Any value (ignored by receiver)
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Value from request
Options	UDINT	0 [a]

[a] For backwards compatibility, no options shall be defined for this command. The receiver shall discard packets with a non-zero option field.

The data portion of the reply contains an array of items providing interface information. It is structured as a Common Packet Format, which contains a 2-octet item count followed by an array of items.

At present no public items are defined for the ListInterfaces reply. If no items are included, the Item Count shall be set to 0.

Some legacy devices may return “Reserved for legacy usage items” (see 4.3.3). Such items shall be ignored unless the receiving device has explicit knowledge of the legacy format and usage.

4.3.2.7 SendRRData

A SendRRData request shall transfer an encapsulated request/reply packet between the originator and target, where the originator initiates the command. The actual request/reply packets shall be encapsulated in the data portion of the message and shall be the responsibility of the target and originator.

NOTE When used to encapsulate Type 2 PDU's, the SendRRData request and reply are used to send encapsulated UCMM messages (see Clause 11).

The SendRRData request encapsulation header shall be as shown in Table 202.

Table 202 – SendRRData request encapsulation header

Field name	Data type	Field value
Command	UINT	SendRRData (0x6F)
Length	UINT	Length of the command specific data
Session handle	UDINT	Session handle
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Chosen by sender
Options	UDINT	0 [a]

[a] For backwards compatibility, no options shall be defined for this command. The receiver shall discard packets with a non-zero option field.

The data portion of the message shall contain request parameters and the encapsulated protocol packet as shown in Table 203.

Table 203 – SendRRData request data portion

Field name	Type	Description
Interface handle	UDINT	Shall be 0
Timeout	UINT	Operation Timeout (0 to 65 535)
Encapsulated protocol packet	ARRAY of octets	See Common Packet Format specification in 4.3.3

The Interface handle shall identify the Communications Interface to which the request is directed. This handle shall be 0 for encapsulating Type 2 PDUs.

The target shall abort the requested operation after the timeout expires. When the “Timeout” field is in the range 1 to 65 535, the timeout shall be set to this number of seconds. When the “Timeout” field is set to 0, the encapsulation protocol shall not have its own timeout. Instead, it shall rely on the timeout mechanism of the encapsulated protocol.

When the SendRRData command is used to encapsulate Type 2 PDU's, the Timeout field shall be set to 0, and shall be ignored by the target.

The encapsulated protocol packet shall be encoded in a Common Packet Format as shown in 4.3.3.

The SendRRData reply, as shown in Table 204, shall contain data in response to the SendRRData request. The reply to the original encapsulated protocol request shall be contained in the data portion of the SendRRData reply.

Table 204 – SendRRData reply encapsulation header

Field name	Data type	Field value
Command	UINT	SendRRData (0x6F)
Length	UINT	Length of the command specific data
Session handle	UDINT	Value from request
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Value from request
Options	UDINT	0 [a]

[a] For backwards compatibility, no options shall be defined for this command. The receiver shall discard packets with a non-zero option field.

The format of the data portion of the reply message shall be the same as that of the SendRRData request message. Since the request and reply share a common format, the reply message contains a Timeout field; however, it is not used.

4.3.2.8 SendUnitData

The SendUnitData request shall send encapsulated connected messages. A reply need not be returned. This command may be used when the encapsulated protocol has its own underlying end-to-end transport mechanism. The SendUnitData request may be sent by either end of the TCP connection.

NOTE When used to encapsulate Type 2 PDU's, the SendUnitData command is used to send Type 2 connected data in both the O⇒T and T⇒O directions.

The format of the SendUnitData request shall be as shown in Table 205.

Table 205 – SendUnitData request encapsulation header

Field name	Data type	Field value
Command	UINT	SendUnitData (0x70)
Length	UINT	Length of data portion
Session handle	UDINT	Session handle
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Any value (ignored by target)
Options	UDINT	0 [a]

[a] For backwards compatibility, no options shall be defined for this command. The receiver shall discard packets with a non-zero option field.

The data portion of the message shall contain request parameters as well as the encapsulated protocol packet as shown in Table 206.

Table 206 – SendUnitData request data portion

Field name	Type	Description
Interface handle	UDINT	Shall be 0
Timeout	UINT	Operation timeout (shall be 0)
Encapsulated protocol packet	ARRAY of octets	See Common Packet Format specification in 4.3.3

Interface handle and Timeout shall be set to zero. The timeout field is not used since no reply is generated upon receipt of a SendUnitData command.

4.3.3 Common packet format

4.3.3.1 General

The common packet format (CPF) defines a standard format for protocol packets that are transported with the encapsulation protocol. The common packet format is a general-purpose mechanism designed to accommodate future packet or address types.

The common packet format shall consist of an item count, followed by a number of items (see Table 207). Some items are classified as “address items” (carries addressing information) or “data items” (carries encapsulated data). The number of items to be included depends on the encapsulation command and usage of the command. Subclause 4.3.3.4 specifies the valid Common Packet Format items for the various commands and usages.

Table 207 – Common packet format

Field name	Data type	Description
Item count	UINT	Number of items to follow
Item #1	Item Struct (see below)	First CPF item
Item #2	Item Struct (see below)	Second CPF item
…	…	…
Item #n	Item Struct (see below)	n^{th} CPF item

The address and data item structure shall be as shown in Table 208.

Table 208 – CPF item format

Field name	Type	Description
Type ID	UINT	Type of item encapsulated
Length	UINT	Length in octets of data to follow
Data	Variable	The data (if length >0)

The item type ID numbers shall be as shown in Table 209.

Table 209 – Item Type ID numbers

Item ID number	Item type	Description
0x0000	address	Null (used for UCMM messages). Indicates that encapsulation routing is NOT needed. Target is either local (Ethernet) or routing info is in a data item.
0x0001 to 0x000B		Reserved for legacy usage[a]
0x000C		ListIdentity response
0x000D to 0x0085		Reserved for legacy usage[a]
0x0086 to 0x0090		Reserved for future extensions[b]
0x0091		Reserved for legacy usage[a]
0x0092 to 0x00A0		Reserved for future extensions[b]
0xA1	address	Connection-based (used for connected messages)
0x00A2 to 0x00A4		Reserved for legacy usage[a]
0x00A5 to 0x00B0		Reserved for future extensions[b]
0x00B1	data	Connected Transport packet
0x00B2	data	Unconnected message
0x00B3 to 0x00FF		Reserved for future extensions[b]
0x0100		ListServices response
0x0101 to 0x010F		Reserved for legacy usage[a]
0x0110 to 0x7FFF		Reserved for future extensions[b]
0x8000	data	Sockaddr Info, originator-to-target
0x8001	data	Sockaddr Info, target-to-originator
0x8002		Sequenced Address item
0x8003 to 0xFFFF		Reserved for future extensions[b]

[a] Items marked as “Reserved for legacy usage” indicate Item IDs that were defined prior to the publication of this standard. Their behavior is undefined in this standard. Devices shall not use these Item IDs without prior knowledge of the legacy usage.

[b] Products compliant with this specification shall not use command codes in this range

4.3.3.2 Address items

4.3.3.2.1 Null

The null address item shall contain only the type id and the length as shown in Table 210. The length shall be zero. No data shall follow the length. Since the null address item contains no routing information, it shall be used when the protocol packet itself contains any necessary routing information. The null address item shall be used for Unconnected Messages.

Table 210 – Null address item

Field name	Data type	Field value
Type ID	UINT	0
Length	UINT	0

4.3.3.2.2 Connected

This address item shall be used when the encapsulated protocol is connection-oriented. The data shall contain a connection identifier, as shown in Table 211.

NOTE Connection identifiers are exchanged in the Forward_Open service of the Connection Manager.

Table 211 – Connected address item

Field name	Data type	Field value
Type ID	UINT	0xA1
Length	UINT	4
Data	UDINT	Connection Identifier

4.3.3.2.3 Sequenced address item

This address item shall be used for class 0 and class 1 connected data. The data shall contain a connection identifier and a sequence number, as shown in Table 212. Usage is further described in 11.3.3.

Table 212 – Sequenced address item

Field name	Type	Field value
Type ID	UINT	0x8002
Length	UINT	8
Data	UDINT	Connection Identifier
	UDINT	Sequence Number

4.3.3.3 Data items

4.3.3.3.1 Unconnected

The data item that encapsulates an unconnected message shall be as shown in Table 213.

Table 213 – Unconnected data item

Field name	Type	Field value
Type ID	UINT	0xB2
Length	UINT	Length, in octets, of the unconnected message
Data	Variable	The unconnected message

The format of the “data” field is dependent on the encapsulated protocol. When used to encapsulate Type 2 PDU's, the format of the “data” field shall be the same as PDU destined for the Message Router. The UCMM header, defined in 4.1.3, shall not be present. The context field in the encapsulation header shall be used for unconnected request/reply matching.

4.3.3.3.2 Connected

The data item that encapsulates a connected transport PDU shall be as shown in Table 214.

Table 214 – Connected data item

Field name	Type	Field value
Type ID	UINT	0xB1
Length	UINT	Length, in octets, of the transport PDU
Data	Variable	The transport PDU

The format of the “data” field is dependent on the encapsulated protocol. When used to encapsulate Type 2 PDU's, the format of the “data” field shall be the same as the connected transport PDU.

4.3.3.3.3 Sockaddr info

The Sockaddr Info items shall be used to communicate IP address or port information necessary to create Class 0 or Class 1 connections. There are separate items for originator-to-target and target-to-originator socket information. The items are present as additional data in Forward_Open / Large_Forward_Open request and reply services encapsulated in a SendRRData message. Subclause 11.2.4 specifies the usage of these items in the context of creating CIP connections.

The Sockaddr Info items shall have the structure shown in Table 215.

Table 215 – Sockaddr info items

Field name	Type	Field value
Type ID	UINT	0x8000 for O⇒T, 0x8001 for T⇒O
Length	UINT	16 (octets)
sin_family	INT	Shall be AF_INET = 2. This field shall be sent in big endian order
sin_port	UINT	For point-point connections, sin_port shall be set to the UDP port to which packets for this Type 2 connection will be sent. For point-point connections, it is recommended that the registered UDP port (0x8AE) be used. When used with a multicast connection, the sin_port field shall be set to the registered UDP port number (0x08AE) and treated by the receiver as “don’t care”. This field shall be sent in big endian order
sin_addr	UDINT	For multicast connections, sin_addr shall be set to the IP multicast address to which packets for this Type 2 connection will be sent. When used with a point-point connection, the sin_addr field shall be treated by the receiver as “don’t care”. It is recommended that the sender set sin_addr to 0 for point-point connections. This field shall be sent in big endian order.
sin_zero	ARRAY of 8 USINT	Recommended value of zero; not enforced

Theformat of the Sockaddr Info item has been patterned after the sockaddr_in structure as defined in the Winsock specification, version 1.1.

4.3.3.4 Valid Common Packet Format item usage summary

The Common Packet Format is used with the following encapsulation commands:

- ListIdentity reply
- ListInterfaces reply
- ListServices reply
- SendRRData request and reply
- SendUnitData
- Transport class 0 and class 1 packets (no encapsulation header)

Table 216 shows the valid usage of CPF items for the various encapsulation commands.

NOTE Clause 11 contains the detailed formats and usage for Type 2 connected and unconnected messages.

Table 216 – Usage of CPF items

Command	Required CPF items	Optional CPF iItems	Action if unexpected or undefined CPF items are present
ListIdentity reply	ListIdentity reply item	None	Ignore items
ListInterfaces reply	None	Legacy devices may return "Reserved for legacy usage" items	Ignore items
ListServices reply	ListServices item for the “Communications” service	Legacy devices may return "Reserved for legacy usage" items	Ignore items
SendRRData request	Address item followed by Data item	When used to encapsulate the Forward_Open service, additional Sockaddr_info items may be present, as specified in Clause 11	Return error (0x0003)
SendRRData reply	Address item followed by Data item	When used to encapsulate the Forward_Open reply, additional Sockaddr_info items may be present, as specified in Clause 11	Signal error to the calling application. For Forward_Open reply, connection shall not be established at the originator
SendUnitData	Address item followed by Data item	None	Discard message
Class 0/1 packet	Address item followed by Data item	None	Discard message

5 Transfer syntax

5.1 Compact encoding

5.1.1 Encoding rules

Subclause 5.1 describes the means by which the data types defined in this standard shall be encoded/transferred across the control network. The abstract syntax definition along with a particular set of encoding rules shall result in the transfer syntax. For application user data, a single set of encoding rules shall be defined (Compact Encoding), resulting in the Compact transfer syntax.

Compact Encoding rules shall start with the encoding rules defined in ISO/IEC 8825-1. Compact Encoding then applies optimization rules, starting with the outer most Service Data Unit (SDU) and progressing to each successive encapsulated SDU. Compact Encoding shall

define a more efficient encoding mechanism by reducing the amount of information (overhead) transferred between devices.

The difference between a Compact encoded value and an ASN.1 encoded value shall be the removal of the fields describing the type and length of the information. The TAG and LENGTH components of an ASN.1-encoded value shall not be transmitted on the control network. In addition, the Compact Encoding rules shall indicate that octet ordering rules are the reverse of those seen in ASN.1.

Given the conditions listed in 5.1.2, general rules shall be applied to an ASN.1 encoded value to generate a Compact encoded value. The general rules shall be as follows:

- remove the Identifier Octets;
- remove the “TAG” octets specified by ASN.1;
- remove the Length Octets;
- remove the “LENGTH” octets specified by ASN.1;
- reverse the octet ordering for multiple content octets.

5.1.2 Encoding constraints

The representation of a variable value using Compact Encoding shall be possible with the following restrictions:

- the variable type shall be fixed length and shall have no conditional or optional fields;
- the encoding of a given variable shall be represented with a constant number of octets derived from the type specification of this variable.

5.1.3 Examples

5.1.3.1 BOOL encoding

The BOOLEAN encoding shall be performed on a single OCTET as described in Table 217.

Table 217 – BOOLEAN encoding

If the value is:	Then:
FALSE	bit 0 of the octet is 0 (’00’H)
TRUE	bit 0 of the octet is 1 (’01’H)

A FALSE BOOL shall be represented as shown in Table 218.

Table 218 – Example compact encoding of a BOOL value

Octet number	1st
BOOL	00

5.1.3.2 SignedInteger encoding

The SignedInteger encoding shall be performed as described in Table 219.

Table 219 – Encoding of SignedInteger values

Octet number	1st	2nd	3rd	4th	5th	6th	7th	8th
SINT	0LSB							
INT	0LSB	1LSB						
DINT	0LSB	1LSB	2LSB	3LSB				
LINT	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB
NOTE The example in Table 220 illustrates the encoding of a variable of type DINT whose value is 0x12345678.								

Table 220 – Example compact encoding of a SignedInteger value

Octet number	1st	2nd	3rd	4th
DINT	78	56	34	12

5.1.3.3 UnsignedInteger encoding

The UnsignedInteger encoding shall be performed as described in Table 221.

Table 221 – UnsignedInteger values

Octet number	1st	2nd	3rd	4th	5th	6th	7th	8th
USINT	0LSB							
UINT	0LSB	1LSB						
UDINT	0LSB	1LSB	2LSB	3LSB				
ULINT	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB
NOTE Table 222 illustrates the encoding of a variable of type UDINT whose value is 0xAABBCCDD.								

Table 222 – Example compact encoding of an UnsignedInteger

Octet number	1st	2nd	3rd	4th
UDINT	DD	CC	BB	AA

5.1.3.4 FixedLengthReal encoding

The FixedLengthReal encoding shall be performed as described in Table 223.

Table 223 – FixedLengthReal values

Octet number	1st	2nd	3rd	4th	5th	6th	7th	8th
REAL	0LSB	1LSB	2LSB	3LSB				
LREAL	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

Table 224 illustrates the encoding of a variable (Float1) whose type is REAL and whose value is Float1: = 10,0.

NOTE 1 The assignment of the value is using the IEC 61131-3 notation. The ASN.1 value is {'41200000'H} in IEEE format: 1,25*23 , exponent is 130 (bias 127), fraction is 25.

Table 224 – Example compact encoding of a REAL value

Octet contents	0LSB	1LSB	2LSB	3LSB
REAL	00	00	20	41

Table 225 illustrates the encoding of a variable (Float2) whose type is LREAL and whose value is Float2: = –100,0.

NOTE 2 The ASN.1 value is {'C059000000000000'H} in IEEE format: 1,562 5*26, exponent is 1 029 (bias 1 023), fraction is 0,562 5.

Table 225 – Example compact encoding of a LREAL value

Octet contents	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB
LREAL	00	00	00	00	00	00	59	C0

5.1.3.5 Time encodings

Table 226 illustrates the encoding of Real numbers with fixedlengths.

Table 226 – FixedLengthReal values

Octet number	1st	2nd	3rd	4th	5th	6th	7th	8th
TIME	0LSB	1LSB	2LSB	3LSB				
DATE	0LSB	1LSB						
TIME_OF_DAY	0LSB	1LSB	2LSB	3LSB				
DATE_AND_TIME	0LSB-Time	1LSB-Time	2LSB-Time	3LSB-Time	0LSB-Date	1LSB-Date		
FTIME	0LSB	1LSB	2LSB	3LSB				
LTIME	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

5.1.3.6 String encodings

Subclause 5.1.3.6 gives examples of the Compact Encoding of STRING, STRING2, STRINGN, and SHORT_STRING data values.

NOTE The preferred string type for user supplied string data is STRING2 due to international character string requirements.

Table 227 provides a generic illustration of the encoding of a STRING value.

Table 227 – STRING value

	Contents (charcount)		Contents (string contents)
STRING	0LSB	1LSB	0LSB

1 octet characters

Table 228 provides a generic illustration of the encoding of a STRING2 value.

Table 228 – STRING2 value

	Contents (charcount)		Contents (string2contents)	
STRING2	0LSB	1LSB	0LSB	1LSB
			2 octet characters	

Table 229 provides a generic illustration of the encoding of a STRINGN value.

Table 229 – STRINGN value

	Contents (charsize)		Contents (charcount)		Contents (stringNcontents)	
STRINGN	0LSB	1LSB	0LSB	1LSB	0LSB	NLSB
					N octet characters	

Table 230 provides a generic illustration of the encoding of a SHORT_STRING value.

Table 230 – SHORT_STRING value

	Contents (charcount)	Contents (short_string)
SHORT_STRING	0LSB	0LSB
		1 octet characters

Table 231 illustrates the encoding of a string variable whose contents equal ”Mill”. Encoding examples of all string types are presented. Character coding is specified in ISO/IEC 10646. The hexadecimal equivalent is: {’4D696C6C’H} for 8 bit encoding.

Table 231 encodes ”Mill” as a STRING type.

Table 231 – Example compact encoding of a STRING value

	Contents (charcount)		Contents (string contents)			
STRING	04	00	4D	69	6C	6C

Table 232 encodes ”Mill” as a STRING2 type.

Table 232 – Example compact encoding of STRING2 value

	Contents (charcount)		Contents (string2 contents)							
STRING2	04	00	4D	00	69	00	6C	00	6C	00

Table 233 encodes ”Mill” as a SHORT_STRING type.

Table 233 – SHORT_STRING type

	Contents (charcount)	Contents (short_string contents)			
SHORT_STRING	04	4D	69	6C	6C

5.1.3.7 FixedLengthBitString encoding

Subclause 5.1.3.7 provides examples of the Compact Encoding of SWORD, WORD, DWORD, LWORD data values. Figure 16 illustrates the bit placement rules associated with the Compact Encoding of a FixedLengthBitString.

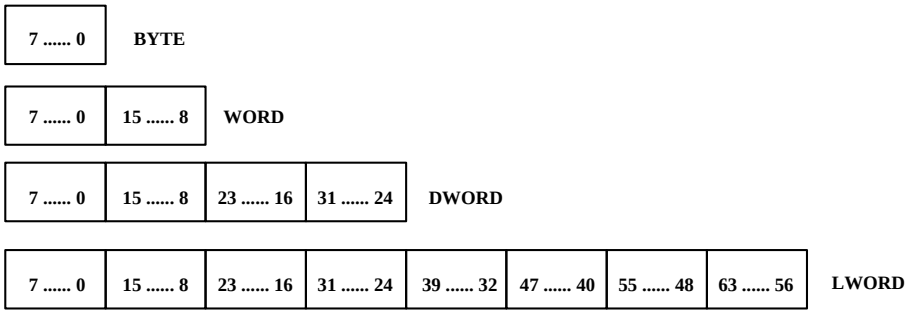


Figure 16 – FixedLengthBitString compact encoding bit placement rules

Figure 17 through Figure 20 illustrate the encoding of SWORD, WORD, DWORD, and LWORD.

Bits In Memory:
7 0
00001111

Compact Encoded BYTE
00001111 or 0x0F

Figure 17 – Example compact encoding of a SWORD FixedLengthBitString

Bits In Memory:
15 0
00001111 11001111

Compact Encoded WORD
11001111 00001111 or 0xCF0F

Figure 18 – Example compact encoding of a WORD FixedLengthBitString

Bits In Memory:
31 0
00001111 11001111 10110110 00111110

Compact Encoded DWORD
00111110 10110110 11001111 00001111 or 0x3EB6CF0F

Figure 19 – Example compact encoding of a DWORD FixedLengthBitString

Bits In Memory:
63 0
11110000 11001111 10110110 00111110 11110000 00101101 00011110 00001111

Compact Encoded LWORD
00001111 00011110 00101101 11110000 00111110 10110110 11001111 11110000 or 0x0F1E2DF03EB6CFF0

Figure 20 – Example compact encoding of a LWORD FixedLengthBitString

5.1.3.8 Array encoding

5.1.3.8.1 One-dimensional array encoding

The Array encoding shall use the encoding rules for the data types for each element and concatenates the elements which compose the array. The encoded values of the array elements shall be encoded in the same order as they are declared in the corresponding ASN.1 type or variable specification. These elements may be of any data type.

The ASN.1–style definition of a single–dimensional array in the control network shall be:

```
ARRAY::= SEQUENCE OF { array_dimension_low_bound,
            array_dimension_high_bound, NetworkData }

Assume the following array definition::
ARRAY1::= SEQUENCE OF {array_dimension_low_bound:= 0,
          array_dimension_high_bound:= 1, UINT}
```

Plugging the UINT values {1,2} into this array definition yields the encoding specified in Table 234.

Table 234 – Example compact encoding of a single dimensional ARRAY

Octet number	1st	2nd	3rd	4th
ARRAY	01	00	02	00

5.1.3.8.2 Two-dimensional array encoding

```
ARRAY::= SEQUENCE OF { array_dimension_low_bound,
array_dimension_high_bound,
     SEQUENCE OF { array_dimension_low_bound,
array_dimension_high_bound, NetworkData } }
```

5.1.3.8.3 Three-dimensional array encoding

```
ARRAY::= SEQUENCE OF { array_dimension_low_bound,
   array_dimension_high_bound,
   SEQUENCE OF { array_dimension_low_bound,
   array_dimension_high_bound,
   SEQUENCE OF { array_dimension_low_bound,
    array_dimension_high_bound, NetworkData } } }
```

Since control network data may comprise either ElementaryData or DerivedData, a new type or variable specification may be required before transmitting the values for the ARRAY.

A multi–dimensional array shall be seen on the wire as a single–dimensional array. The order of the array elements shall be maintained via the packing/unpacking sequence followed by the end nodes. The sequence followed shall be to access the Nth dimension of the array for all values of the other dimensions.

This shall be achieved by first accessing the array with all dimensions set to their initial index values. After this the Nth dimension is incremented through all of its index values. When the end of the index range for the Nth dimension is reached, the (N-1)th dimension is incremented, and the Nth dimension is set to its initial index value. This process is repeated until all of the array's dimensions have reached the ends of their index ranges, and results in the array being packed into the message buffer as a single–dimensional array. The same procedure is followed to unpack the single–dimensional array into a multi–dimensional array.

The two–dimensional array

```
ARRAY1 [0..1 , 0..2] of UINT:= { { 1, 2, 3 },
                                 { 4, 5, 6 } }
```

results in the data stream shown in Table 235 when it is packed into a single–dimensional array following the compact encoding rules:

Table 235 – Example compact encoding of a multi-dimensional ARRAY

Octet number	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th
ARRAY	01	00	02	00	03	00	04	00	05	00	06	00

5.1.3.9 Structure encoding

The structure encoding shall use the encoding rules for the data types for each element and concatenates the elements which compose the structure.

The encoded values of the structure elements shall be encoded in the same order as they are declared in the corresponding ASN.1 type or variable specification. These elements may be of any Data type.

STRUCT::= SEQUENCE OF NetworkData –- May be different types

Since NetworkData may be comprised of either ElementaryData or DerivedData, a new type or variable specification may be required before transmitting the values for the STRUCT.

Assume the following structure definition:

newStruct::= SEQUENCE { BOOL,UINT,DINT}

Plugging the values {TRUE,'1234'H,'56789ABC'H} into the structure results in the encoding as specified in Table 236.

Table 236 – Example compact encoding of a STRUCTURE

Octet number	1st	2nd	3rd	4th	5th	6th	7th
newStruct	01	34	12	BC	9A	78	56

5.1.3.10 Complex encoded data format examples

5.1.3.10.1 General

The examples 5.1.3.10.2 and 5.1.3.10.3 show how more complex data formats shall be packed. Example 5.1.3.10.2 shows the packing of an array of structures. Example 5.1.3.10.3 shows how a structure with an array element is packed.

5.1.3.10.2 Example 1: encoding an array of structures:

```
STRUCT1::= SEQUENCE OF {
            UINT     ele1;
            USINT    ele2;
            USINT    ele3;
            USINT    ele4;
            UINT     ele5
}
```

```
ARRAY1 [ 0..1 , 0..2 ] of STRUCT1:= {
          { { 1, 2, 3, 4, 5 }, { 6, 7, 8, 9, 10 },
           { 11, 12, 13, 14, 15 } },
          { { 15, 14, 13, 12, 11 }, { 10, 9, 8, 7, 6 },
           { 5, 4, 3, 2, 1 } } }
```

results in the following data stream:

```
[01][00][02][03][04][05][00] [06][00][07][08][09][0A][00]
[0B][00][0C][0D][0E][0F][00] [0F][00][0E][0D][0C][0B][00]
[0A][00][09][08][07][06][00] [05][00][04][03][02][01][00]
```

5.1.3.10.3 Example 2: encoding a structure with an array element

```
STRUCT2::= SEQUENCE OF {
            UINT     ele1;
            ARRAY [ 0..2 ] of USINT array2;
            UINT     ele5;
}
STRUCT2:= { 1, { 2, 3, 4 }, 5 }
```

results in the following data stream:

```
[01][00] [02][03][04] [05][00]
```

5.2 Data type reporting

5.2.1 Object data representation

Objects may choose to implement a mechanism for reporting Data Type of a particular Attribute or transmitting type information along with the actual data. Subclause 5.2 defines the means by which Data Typing information is conveyed.

The specification of Data Type information utilizes the ASN.1 methodology specified in ISO/IEC 8824-1 and ISO/IEC 8825-1 with the control network defined optimizations to encode the **DataTypeSpecification** production defined in 4.2.4.

The control network defined optimization is that the Length Octet of a NULL type is not encoded.

NOTE For example, the encoding of 'abc [PRIVATE 1] IMPLICIT NULL' would be: 0xC1 (a tag with no length octet).

5.2.2 Elementary data type reporting

Elementary data types shall be identified using the identification codes defined in Table 237. These codes shall define the encoding of the primitive members of the DataTypeSpecification production. The control network specifies that ASN.1 NULL types shall not report the Length Octet of zero (0).

Table 237 – Identification codes and descriptions of elementary data types

Data type name	Data type code (in hex)	Data type description
BOOL	C1	Logical Boolean with values TRUE and FALSE
SINT	C2	Signed 8–bit integer value
INT	C3	Signed 16–bit integer value
DINT	C4	Signed 32–bit integer value
LINT	C5	Signed 64–bit integer value
USINT	C6	Unsigned 8–bit integer value

Data type name	Data type code (in hex)	Data type description
UINT	C7	Unsigned 16–bit integer value
UDINT	C8	Unsigned 32–bit integer value
ULINT	C9	Unsigned 64–bit integer value
REAL	CA	32–bit floating point value
LREAL	CB	64–bit floating point value
STIME	CC	Synchronous time information
DATE	CD	Date information
TIME_OF_DAY	CE	Time of day
DATE_AND_TIME	CF	Date and time of day
STRING	D0	character string (1 octet per character)
SWORD	D1	bit string - 8-bits
WORD	D2	bit string - 16-bits
DWORD	D3	bit string - 32-bits
LWORD	D4	bit string - 64-bits
STRING2	D5	character string (2 octets per character)
FTIME	D6	Duration (high resolution)
LTIME	D7	Duration (long)
ITIME	D8	Duration (short)
STRINGN	D9	character string (N octets per character)
SHORT_STRING	DA	character sting (1 octet per character, 1 octet length indicator)
TIME	DB	Duration (milliseconds)
EPATH	DC	Path segments
STRINGI	DE	International Character String

5.2.3 Constructed data type reporting

5.2.3.1 General

Subclause 5.2.3 details the means by which the structure and array information presented within the DataTypeSpecification production is represented.

Table 238 specifies the identification codes used for the various constructed data types.

Table 238 – Identification codes and descriptions of constructed data types

Data type name	Data type code (in hex)	Data type description
Abbreviated Structure	A0	Structure identified by 16-bit CRC value
Abbreviated Array	A1	Array identified by 16-bit CRC value
Formal Structure	A2	Structure defined by member type list
Formal Array	A3	Array defined by type and size
	A4-A7	Reserved – do not use
Formal Handle Structure	A8	Structure defined by member type and member handle list
	A9-BF	Reserved

5.2.3.2 Structure type definition

5.2.3.2.1 General

The control network defines three different methods for reporting Structure type definitions:

- Formal Encoding (FormalStrucTypeSpec);
- Formal Handle Encoding (FormalHandleStrucTypeSpec);
- Abbreviated Encoding (AbbreviatedStrucTypeSpec).

Formal encoding shall be used to provide a detailed report of the complete structure definition, including the complete definition of all component data types(the structure member number can be obtained based on the component position). Formal Handle encoding is used to provide a detailed report of the complete structure definition, including the complete definition of all component data types and component handles. Abbreviated encoding shall be used to specify a shorter form of the structure definition. This shorter form shall not include the data types associated with the structure's components.

5.2.3.2.2 Formal encoding for structure type information

Table 239 defines the formal encoding format for non-nested structures.

Table 239 – Formal structure encoding definition

Octet	Definition
0	Formal Structure (A2)
1	Length of structure definition in octets (n)
2 to (n+1)	Data type code (C0 to DF)

The examples in 5.2.3.2.3 and 5.2.3.2.4 illustrate formal encoding for structure type specifications. This is actually an example of the encoding of the FormalStrucTypeSpec production defined in 4.2.4.

5.2.3.2.3 Example 1

Figure 21illustrates the encoding of the following structure definition.

```
STRUCT::= SEQUENCE OF { BOOL, UINT, DINT }
```

STRUCT type	Type length	Data type (BOOL)	Data type (UINT)	Data type (DINT)
A2	03	C1	C7	C4
NOTE The IMPLICIT NULL types from the DataTypeSpecification production are not followed by a Length Octet of zero (0).				

Figure 21 – Example 1 of formal encoding of a structure type specification

5.2.3.2.4 Example 2

Figure 22 illustrates the encoding of the following structure definition.

```
STRUCT_MAIN::= SEQUENCE OF { UINT, STRUCT_SUB, INT }

with subelement STRUCT_SUB defined as:

STRUCT_SUB::= SEQUENCE OF { UINT, SINT, INT }
```

Struct type	Type length	Component						
		Data type (UINT)	Struct type	Type length	Nested Component			
					Data type (UINT)	Data type (SINT)	Data type (INT)	Data type (INT)
A2	07	C7	A2	03	C7	C2	C3	C3

Figure 22 – Example 2 of formal encoding of a structure type specification

5.2.3.2.5 Formal Encoding for Handle Structure Type Information

Table 240 defines the formal encoding format for structures with handles.

Table 240 – Formal structure with handles encoding definition

Octet	Definition
0	Formal Handle Structure (A8)
1	Length of structure definition in octets (n)
2	Handle length (1, 2 or 4 octets)
3	Data type code (C0 to DF) 1
4 to 4 + (Handle length – 1)	Handle (1, 2 or 4 octet value assigned by application) [a]
[a] Additional members are represented by pairs of data type codes and handles.	

The two examples in 5.2.3.2.6 and 5.2.3.2.7 illustrate formal encoding for handle structure type specifications. This is actually an example of the encoding of the FormalHandleStrucTypeSpec production defined in 4.2.4.

5.2.3.2.6 Example 3

Figure 23 shows the encoding of the following structure definition.

```
STRUCT::= SEQUENCE OF { BOOL, UINT, DINT }
```

Struc type	Type length	Handle length	Data type (BOOL)	Member handle	Data type (UINT)	Member handle	Data type (DINT)	Member handle
A8	10	04	C1	D9 B9 74 3E	C7	F4 15 59 93	C4	0B 7B 24 A6

Figure 23 – Example 3 of formal encoding of a handle structure type specification

5.2.3.2.7 Example 4

Figure 24 shows the encoding of the following structure definition

```
STRUCT_MAIN::= SEQUENCE OF { UINT, STRUCT_SUB, INT }
```

Struct type	Type length	Handle length	Component										
			Data type (UINT)	Member handle	Struct type	Type length	Handle length	Nested component				Data Type (INT)	Handle
								Data Type (SINT)	Member Handle	Data Type (INT)	Member Handle		
A8	0E	02	C7	90 1D	A8	05	01	C2	F8	C3	B2	C3	36 ED

Figure 24 – Example 4 of formal encoding of a handle structure type specification

5.2.3.2.8 Abbreviated encoding for structure type information

Table 241 defines the abbreviated encoding format for structures.

Table 241 – Abbreviated structure encoding definition

Octet	Definition
0	Abbreviated Structure (A0)
1	Length of abbreviated structure (02)
2 to 3	UINT containing CRC

The example in 5.2.3.2.9 illustrates the abbreviated encoding for structure type specifications. This is actually an example of the encoding of the AbbreviatedStrucTypeSpec production defined in 4.2.4.

The UINT defined within the AbbreviatedStrucTypeSpec production shall be initialized with a 16 bit Cyclic Redundancy Check (CRC) value (see IEC 61158-4-2). This can be used by application logic to determine whether or not the format of the structure has changed. The octet stream used to produce the CRC is the actual formally encoded (FormalStrucTypeSpec) structure type specification.

5.2.3.2.9 Example 5

Figure 25 shows the abbreviated encoding of the structure definition presented in 5.2.3.2.8:

Struct Type	Type Length	UINT containing CRC	
A0	02	C7	26

Figure 25 – Example 5 of abbreviated encoding of a structure type specification

NOTE The algorithms presented in this standard are used to generate the CRC value of 0x26C7 from the Formally Encoded type specification: [A2][07][C7][A2][03] [C7][C2][C3][C3].

5.2.3.3 Array type definition

5.2.3.3.1 General

Two different methods for reporting array type definitions shall be:

- Formal Encoding (FormalArrayTypeSpec);

5.2.3.3 Array type definition

5.2.3.3.1 General

Two different methods for reporting array type definitions shall be:

- Formal Encoding (FormalArrayTypeSpec);
- Abbreviated Encoding (AbbreviatedArrayTypeSpec).

Formal encoding shall be used to provide a detailed report of the complete array definition, including the data content and the array's dimensions. Abbreviated encoding shall be used to specify a shorter form of the array definition. This shorter form shall not include information specifying the array's dimensions.

5.2.3.3.2 Formal encoding for array type information

Table 242 defines the formal encoding format for arrays.

Table 242 – Formal array encoding definition

Octet	Definition
0	Formal Array (A3)
1	Length of array definition in octet (s)
2	Lower bound tag (0x80)
3	Lower bound length (n – values 1, 2 or 4)
4 to (4 + (n-1))	Lower bound
(4 + n)	Higher bound tag (0x81)
(5 + n)	Higher bound length (m – values 1, 2 or 4)
(6 + n) to ((6 + n) + (m-1))	Higher bound m
((6 + n) + m) to (6 + (s – 5))	One of: 1 Array element data type code (C0 to DF) 2 Formal array encoding 3 Formal structure encoding 4 Formal handle structure encoding

The examples 5.2.3.3.3 and 5.2.3.3.4 illustrate formal encoding for structure type specifications. This is actually an example of the encoding of the FormalArrayTypeSpec production defined in 4.2.4.

5.2.3.3.3 Example 1

Figure 26 shows the formal encoding of the following array definition.

```
ARRAY1::= SEQUENCE OF { array_dimension_low_bound:= 0,
                        array_dimension_high_bound:= 9,
                        UINT
```

Array	Type length	Lower bound tag	Lower bound length	Lower bound	Upper bound tag	Upper bound length	Upper bound	Data Type (UINT)
A3	07	80	01	00	81	01	09	C7

Figure 26 – Example 1 of formal encoding of an array type specification

NOTE The IMPLICIT NULL types from the DataTypeSpecification production are not followed by a Length Octet of zero (0).

5.2.3.3.4 Example 2

Figure 27 shows the encoding of the following array definition.

```
ARRAY1::= SEQUENCE OF { array_dimension_low_bound:= 0,
                        array_dimension_high_bound:= 19,
                      SEQUENCE OF { array_dimension_low_bound:= 0,
                          array_dimension_high_bound:= 255,
STRUCT_ELE } }
```

in which STRUCT_ELE is defined as:

```
STRUCT_ELE::= SEQUENCE OF { UINT, SINT, INT }
```

Formal Encoding: [A3][13][80][01][00][81][01][13][A3][0B]80][01][00][81][01][FF][A2][03][C7][C2][C3]

Description:

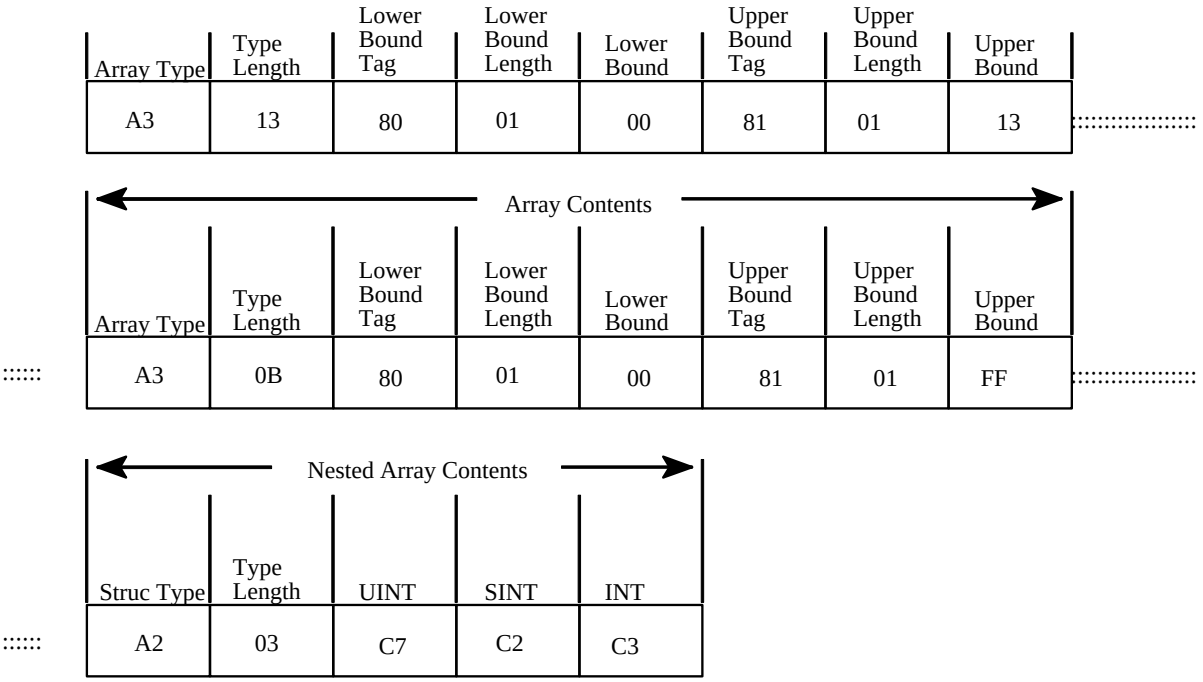


Figure 27 – Example 2 of formal encoding of an array type specification

NOTE The IMPLICIT NULL types from the DataTypeSpecification production are not followed by a Length Octet of zero (0).

5.2.3.3.5 Abbreviated tag encoding for array type information

Table 243 defines the abbreviated encoding format for arrays.

Table 243 – Abbreviated array encoding definition

Octet	Definition
0	Abbreviated Array (A1)
1	Length of abbreviated array (n)
2 to (2+n-1)	One of: 1 Array element data type code (C0 to DF) 2 Abbreviated array encoding [a] 3 Abbreviated structure encoding
[a] Each dimension of the array shall be represented by an abbreviated array.	

The abbreviated encoding of an Array type shall not include the information specifying the Array's dimensions. This is actually an example of the encoding of the AbbreviatedArrayTypeSpec production defined in 4.2.4.

5.2.3.3.6 Example 1

Figure 28 shows the abbreviated encoding of the following array definition:

```
ARRAY2::= SEQUENCE OF { array_dimension_low_bound:= 0,
                        array_dimension_high_bound:= 9,
                        UINT }
```

Array type	Type length	Data type (UINT)
A1	01	C7

Figure 28 – Example 1 of abbreviated encoding of an array type specification

NOTE The IMPLICIT NULL types from the DataTypeSpecification production are not followed by a Length Octet of zero (0).

5.2.3.3.7 Example 2

Figure 29 shows the abbreviated encoding of the following array definition:

```
ARRAY::= SEQUENCE OF { array_dimension_low_bound:= 0,
                       array_dimension_high_bound:= 19,
                       SEQUENCE OF { array_dimension_low_bound:= 0,
                       array_dimension_high_bound:= 899,
STRUCT_ELE } }
```

in which STRUCT_ELE is defined as:

```
STRUCT_ELE::= SEQUENCE OF { UINT, SINT, INT }
```

Array type	Type length	Array type	Type length	Structure (STRUCT_ELE)			
				Struct type	Type length	UINT containing CRC	
A1	06	A1	04	A0	02	59	51

Figure 29 – Example 2 of abbreviated encoding of an array type specification

NOTE The algorithms presented in this standard are used to generate the CRC value of 0x5159 from the Formally Encoded type specification: [A2][03][C7][C2][C3].

6 Structure of FAL protocol state machines

Interface to FAL services and protocol machines are specified in Clause 6. Conventions used for the descriptions are given in IEC 61158-1.

This fieldbus follows the structure outlined for Type 1 fieldbus with the following specific features:

- There is no formal definition of AP-Context Machine
- There is a formally-defined FSPM Machine serving as an interface between FAL User and ARPM.
- There are ARPM Machines of two different types:

1) one ARPM machine for connection-less application relationships
2) seven ARPM machines for connection-oriented application relationships

- DMPM Machines are defined at the interface to the supported Data-link layer (Type 2 or others).

7 AP-Context state machine

7.1 Overview

Type 2 supports the AP-Context State Machine specified in, when the Connection object is also supported.

7.2 Connection object state machine

7.2.1 I/O Connection instance behavior

Figure 30 provides a general overview of the behavior associated with an **I/O Connection object** (**instance_type** attribute = I/O).

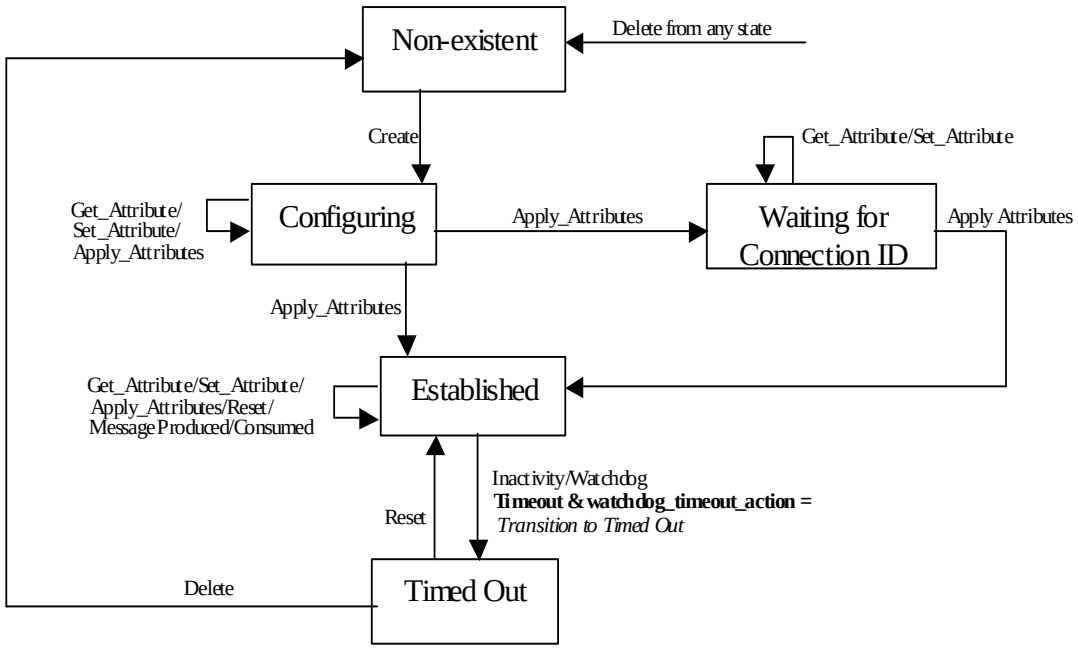


Figure 30 – I/O Connection object state transition diagram

Important: Table 244 provides a detailed State Event Matrix for an I/O Connection object and implementations should be based on this information. This State Event Matrix does not dictate rules with regards to product specific, internal logic. Any attempt to access the Connection Class or a Connection object Instance may need to pass through product specific verification. This may result in an error scenario that is not indicated by the SEM in Table 244. This may also result in additional, product specific indications delivered from a Connection object to the application and/or a specific application object. The point to remember is that the Connection object shall exhibit the externally visible behavior specified by the SEM and the attribute definitions.

Table 244 – I/O Connection state event matrix

Event	I/O Connection object state				
	Non-Existent	**Configuring**	**Waiting for Connection ID**	**Established**	**Timed Out**
Connection Class receives a Create Request	Class instantiates a Connection object. Set instance_type to I/O. Set all other attributes to default values. Transition to Configuring	Not applicable	Not applicable	Not applicable	Not applicable
Connection Class receives a Delete Request	Error: Object does not exist (General Error Code 16_{hex})	Release all associated resources. Transition to Non-existent	Release all associated resources. Transition to Non-existent.	Release all associated resources. Transition to Non-existent.	Release all associated resources. Transition to Non-existent.
Set_Attribute _Single	Error: Object does not exist (General Error Code 16_{hex})	Validate/service the request based on internal logic and per the Access Return response.	If request to modify produced or consumed_connection_ id then validate the value and service the request. Return appropriate response. If this is a request to access an attribute other than the produced or consumed_connection_ id, then return an Error Response whose General Error Code is set to $0C_{hex}$ (The object can not perform the requested service in its current mode/state)	Validate/service the request based on internal logic and per the Access Rules Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules Return appropriate response.
Get_Attribute _Single	Error: Object does not exist (General Error Code 16_{hex})	Validate/service the request based on internal logic and per the Access Rules Return response.	Validate/service the request based on internal logic and per the Access Rules Return response.	Validate/service the request based on internal logic and per the Access Rules Return response.	Validate/service the request based on internal logic and per the Access Rules Return response.
Reset	Error: Object does not exist (General Error Code 16_{hex})	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = $0C_{hex}$)	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = $0C_{hex}$)	Cancel the current Inactivity/Watchdog Timer. Using the value in the expected_packet_rate attribute, re-start the Inactivity/Watchdog Timer. A success response is returned even if an Inactivity/Watchdog Timer is not utilized by the Connection object (Client Transport Class 0, expected_packet_rate = $00C_{hex}$).	Using the value in the expected_packet_rate attribute, start the Inactivity/Watchdog timer and transition back to the Established state. If the expected_packet_rate attribute has been set to zero (0) while the Connection was in the Timed Out state, then just transition back to Established without activating an Inactivity/Watchdog Timer.

Event	I/O Connection object state				
	Non-Existent	**Configuring**	**Waiting for Connection ID**	**Established**	**Timed Out**
Apply_ Attributes	Error: Object does not exist (General Error Code 16_{hex})	Deliver Connection object to the application which validates the attribute information. If either of the connection_id attributes (produced or consumed) needs to be configured and cannot be generated by this module, then perform all the other steps necessary to configure the connection, return a Successful Response and transition to the Waiting For Connection ID state. The inability to generate a produced or consumed connection ID value IS NOT reported as an error. If all attributes are validly configured, then perform all the steps necessary to satisfy this connection, start all required timers, and transition to the Established state. If an error is detected, then an Error Response is returned and the Connection remains in the Configuring state[a] and, if a Client Connection with a production trigger value of 0 or 1 (Cyclic or Change of State), produce initial data.	If either of the connection_id attributes (produced or consumed) still needs to be configured and cannot be generated by this module, then return a Successful Response and remain in the Waiting For Connection ID state. The inability to generate a produced or consumed connection ID value IS NOT reported as an error. If the produced and/or consumed_connection_ id attributes are now validly configured, then transition to the Established state and return a Successful Response[a] and, if a Client Connection with a production trigger value of 0 or 1 (Cyclic or Change of State), produce initial data.	All modifications take place immediately once the Connection has transitioned to the Established state. Return Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = $0C_{hex}$)	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = $0C_{hex}$)
Receive_Data	Not applicable	Discard the message	Discard the message	If a complete, valid[b] message has been received, reset the Inactivity/Watchdog Timer[c] and deliver the I/O Message to the Application. A Connection object shall exhibit the externally visible behavior associated with the current state of its attributes (see Access Rules). If this is a fragmented portion of an I/O Message, process as specified by subnet.	Discard the message

Event	I/O Connection object state				
	Non-Existent	**Configuring**	**Waiting for Connection ID**	**Established**	**Timed Out**
Send_Message	Not applicable	Return internal error - do not send the message	Return internal error - do not send the message	Transmit the complete I/O Message or fragment as required by the subnet. If this is a Client Connection and the expected_packet_rate attribute is non-zero, restart the Transmission Trigger Timer.	Return internal error - do not send the message
Inactivity/ Watchdog Timer expires	Not applicable	Not applicable	Not applicable	Examine the watchdog_timeout_action attribute of the Connection and perform the indicated action. If the watchdog_timeout_action attribute indicates that the Connection is to remain in the Established state (*Auto Reset*), then immediately re-start the Inactivity/Watchdog Timer.	Not applicable

a If the configuration indicates that a Message ID needs to be allocated and an available Message ID does not exist in the specified Message Group, then an Error Response whose General Error Code indicates *Resource Unavailable* (02_{hex}) is returned. If a Connection object attribute value passed the range check when it was initially configured but the attribute value conflicts with another piece of information in the node when the Apply request is processed, then an Error Response is returned whose General Error Code is set to *Invalid Attribute Value* (09_{hex}) and whose Additional Code is set to the Attribute ID of the *offending* Connection object Attribute ID.

b The Connection object verifies that the length of the received I/O Message is less than or equal to the consumed_connection_size attribute prior to processing the message. If the length of the received message is less than or equal to the consumed_connection_size attribute, then the I/O Connection object resets the Inactivity/Watchdog Timer, exhibits the externally visible behavior indicated by its attribute settings, and delivers the message to the Application. If the length of the received message is greater than the consumed_connection_size attribute, then the I/O Connection object immediately discards the message and discontinues any subsequent processing. This is the only message content validation performed by an I/O Connection object. Subsequent validation shall be performed by the Application.

c If a fragmented message is being received, then the Inactivity/Watchdog Timer is not reset until the entire message has been validly received.

NOTE Attribute Access Rules are specified in IEC 61158-5-2, 6.2.3.2.1.4.

Important: The Receive_Data event is only delivered to an I/O Connection when a message whose Connection Identifier Field matches the consumed_connection_id attribute is received. If a message is received whose Connection Identifier Field does not match any Established Connection object's consumed_connection_id attribute, then the message is discarded. Connection Identifiers are subnet-type specific.

If an implementation detects that it does not support an Explicit Messaging Service indicated in Table 244, then an Error Response specifying a General Error Code 08 (Service Not Supported) is returned.

7.2.2 Bridged Connection instance behavior

Figure 31 provides a general overview of the behavior associated with a **Bridged** Connection object (**instance_type** attribute = Bridged). Bridged connections are used to make connections offlink. Both I/O and Explicit Messaging can be accomplished using this

connection type. The Connection Manager object definition provides more details these types of connections.

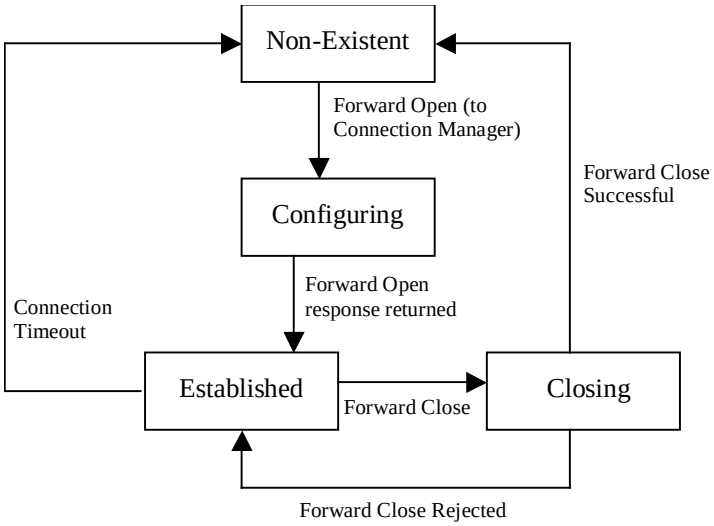


Figure 31 – Bridged Connection object state transition diagram

Table 245 provides a detailed State Event Matrix for a Bridged Connection object.

Table 245 – Bridged Connection state event matrix

Event	Bridged Connection object state			
	Non-Existent	**Configuring**	**Established**	**Closing**
Connection Manager receives a Forward Open Request	Connection Class instantiates a Connection object. Set **instance_type** to **Bridged**. Set attributes to value delivered by Forward Open service or default for Bridged connections. Transition to **Configuring.**	Not applicable	Not applicable	Not applicable
Connection Manager receives notification that connection establishment is complete to target	Not applicable	Transition to **Established**	Not applicable	Not applicable
Connection Manager receives a Forward Close Request	Not applicable	Ignore event	Transition to **Closing**	Ignore event
Connection Manager receives a Forward Close Response	Not applicable	Release all resources and transition to **Non-Existent**	Release all resources and transition to **Non-Existent**	Release all resources and transition to **Non-Existent**
Delete	Error: Object does not exist (General Error Code 16_{hex})	Release all resources and transition to **Non-Existent**	Release all resources and transition to **Non-Existent**	Release all resources and transition to **Non-Existent**

Event	Bridged Connection object state			
	Non-Existent	**Configuring**	**Established**	**Closing**
Get_Attribute_Single	Error: Object does not exist (General Error Code 16_{hex})	Validate/service the request based on internal logic and per the Access Rules. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules. Return appropriate response.
Set_Attribute_Single	Error: Object does not exist (General Error Code 16_{hex})			
Reset	Error: Object does not exist (General Error Code 16_{hex})	Error: Service Not Supported (General Error Code 08_{hex})	Error: Service Not Supported (General Error Code 08_{hex})	Error: Service Not Supported (General Error Code 08_{hex})
Apply_Attributes	Error: Object does not exist (General Error Code 16_{hex})	Error: Service Not Supported (General Error Code 08_{hex})	Error: Service Not Supported (General Error Code 08_{hex})	Error: Service Not Supported (General Error Code 08_{hex})
Receive_Data	Not applicable	Ignore event	Invoke send service of Connection object on destination port, passing the received data.	Ignore event
Send_Message	Not applicable	Ignore event	Send data on subnet.	Ignore event
Inactivity/Watchdog Timer expires	Not applicable	Not applicable	Release all resources and transition to **Non-Existent**	Release all resources and transition to **Non-Existent**
NOTE Attribute Access Rules are specified in IEC 61158-5-2, 6.2.3.2.1.4.				

7.2.3 Explicit Messaging Connection instance behavior

Figure 32 provides a general overview of the behavior associated with an **Explicit Messaging Connection object** (**instance_type** attribute = Explicit Messaging).

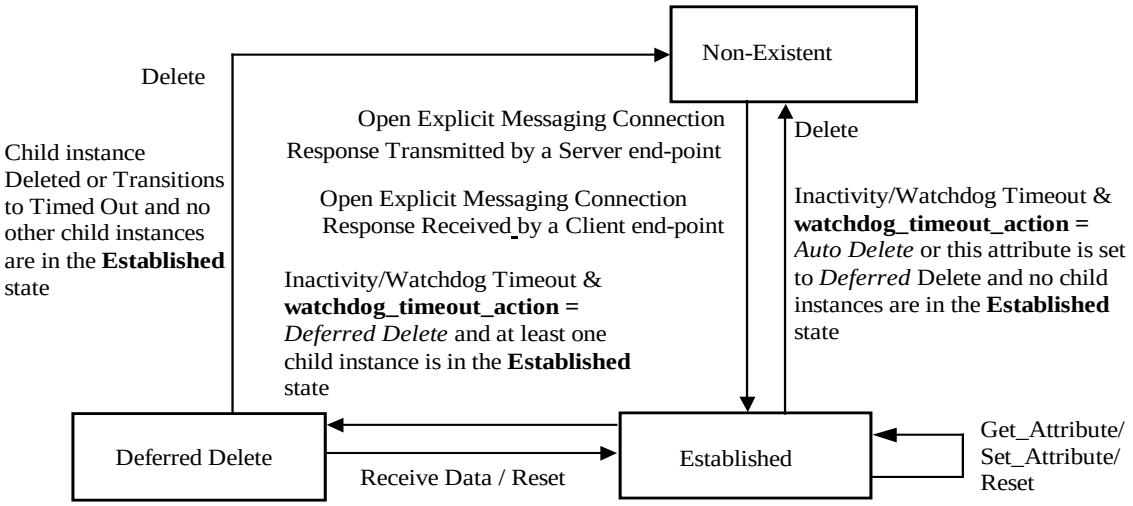


Figure 32 – Explicit Messaging Connection object state transition diagram

Table 246 provides a detailed State Event Matrix for an Explicit Messaging Connection object. Implementations should be based on the information in Table 246.

Table 246 – Explicit Messaging Connection state event matrix

Event	Explicit Messaging Connection object state		
	Non-Existent	**Established**	**Deferred Delete**
UCMM receives an Open Explicit Messaging Connection Request/Response and invokes the Create service of the Connection Class	If possible, Class instantiates Connection object. Set **instance_type** attribute to *Explicit Messaging*[a]. Other attributes are automatically configured using the information in the Open Explicit Messaging Connection Request/Response. Transition to **Established.** If request received, Transmit Open Explicit Messaging Connection Response.	Not applicable	Not applicable
UCMM receives a Close Request and invokes the Delete service of the Connection Class	Error: Object does not exist (General Error Code 16_{hex})	Release all associated resources. Transition to **Non-existent.**	Release all associated resources. Transition to **Non-existent.**
Connection Class receives a Delete Request	Error: Object does not exist (General Error Code 16_{hex})	Release all associated resources. Transition to **Non-existent.**	Release all associated resources. Transition to **Non-existent.**
Set_Attribute_Single	Error: Object does not exist (General Error Code 16_{hex})	Validate/service the request based on internal logic and per the Access Rules. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules. Return appropriate response.
Get_Attribute_Single	Error: Object does not exist (General Error Code 16_{hex})	Validate/service the request based on internal logic and per the Access Rules. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules. Return appropriate response.
Reset	Error: Object does not exist (General Error Code 16_{hex})	Cancel the current Inactivity/Watchdog Timer. Using the value in the **expected_packet_rate** attribute, re-start the Inactivity/Watchdog Timer.	Using the value in the **expected_packet_rate** attribute, re-start the Inactivity/Watchdog Timer and transition back to the Established state.
Apply_Attributes	Error: Object does not exist (General Error Code 16_{hex})	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0Chex)	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0Chex)
Receive_Data	Not applicable	If a valid message or message fragment has been received, then reset the Inactivity/Watchdog Timer[b]. Either process/store the fragment or handle the Explicit Message.	If a valid message or message fragment has been received, then restart the Inactivity/Watchdog Timer[b] and transition back to the **Established** state. Either process/store the fragment or handle the Explicit Message.

Event	Explicit Messaging Connection object state		
	Non-Existent	**Established**	**Deferred Delete**
Send_Message	Not applicable	Examine the length of the message to transmit and, if necessary, perform a fragmented series of transmissions. Otherwise, transmit the complete Explicit Message.	Examine the length of the message to transmit and, if necessary, perform a fragmented series of transmissions. Otherwise, transmit the complete Explicit Message.
Inactivity/Watchdog Timer expires	Not applicable	If the **watchdog_timeout_action** attribute is set to *Auto Delete* or is set to *Deferred Delete* and no child connection instances are in the **Established** state release all associated resources and Transition to **Non-existent**. If the **watchdog_timeout_action** attribute is set to *Deferred Delete* and at least one child connection instance is in the **Established** state transition to **Deferred Delete**	Not applicable.
Child connection instance Deleted or Transitions to Timed Out	Not applicable	Ignore event	If no other child connection instances are in the **Established** state release all associated resources and transition to **Non-existent.**

[a] If the configuration indicates that a connection resource needs to be allocated and an available resource does not exist, then an Error Response whose General Error Code indicates Resource Unavailable (02hex) is returned.

[b] On CP 2/3, the MAC ID field within the Message Header of all Explicit Messages and/or Message Fragments is examined. If the Destination MAC ID is specified in the Connection ID (CAN Identifier Field), then the Source MAC ID of the other end-point shall be specified in the Message Header. If the Source MAC ID is specified in the Connection ID, then the receiving module's MAC ID shall be specified in the Message Header. If either of these checks fail, then the Inactivity/Watchdog Timer IS NOT reset and the message/message fragment is discarded.

NOTE Attribute Access Rules are specified in IEC 61158-5-2, 6.2.3.2.1.4.

Important: The Receive_Data event is only delivered to an Explicit Messaging Connection when a message who's Connection ID matches the consumed_connection_id attribute is received. If a message is received whose connection id does not match any Established Connection object's consumed_connection_id attribute, then the message is discarded.

If an implementation detects that it does not support an Explicit Messaging Service indicated in Table 246, then an Error Response specifying a General Status Code 08 (Service Not Supported) is returned.

8 FAL service protocol machine (FSPM)

8.1 General

This fieldbus FAL Service Protocol State Machine maps FAL User services onto services of communication objects internal to FAL.

8.2 Primitive definitions

The Objects within FSPM shall provide the following services:

Get_Attribute_All	Reads values of all attributes of the specified object class or instance
Set_Attribute_All	Writes specified values to all attributes of the specified object class or instance
Get_Attribute_List	Reads values of the specified list of attributes of the specified object class or instance
Set_Attribute_List	Writes specified values to the specified list of attributes of the specified object class or instance
Get_Attribute_Single	Reads value of the specified attribute of the specified object class or instance
Set_Attribute_Single	Writes specified value to the specified attribute of the specified object class or instance
Reset	Resets the specified object class or instance
Create	Creates an instance of the specified object class
Delete	Deletes an instance of the specified object class
Start	Starts execution of the specified object
Stop	Stops execution of the specified object
Find_Next_Object_Instance	Finds the identifier of the next unused instance of the specified object class
NOP	Triggers corresponding NOP response from the specified object
Apply_Attributes	Causes pending attribute values to become active in the specified object
Save	Saves attributes contents of the specified object
Restore	Restores attributes contents of the specified object
Get_Member	Reads member(s) information from within an attribute
Set_Member	Writes member(s) information in an attribute
Insert_Member	Inserts member(s) into an attribute
Remove_Member	Removes member(s) from an attribute
Group_Sync	Verifies that each member of a group is synchronized to System Time

Refer to IEC 61158-5-2 for detailed definition of these services.

Primitives of Common Services exchanged between UCMM and FAL User are shown in Table 248 and Table 249.

All services have the following common parameters, as shown in Table 247.

Table 247 – Primitives issued by FAL user to FSPM

Common parameters	Req	Ind	Rsp	Cnf
Argument	M			
AREP	M			
Local	S			
UCMM Record identifier	S			
Transport identifier	S			
Receiver/Server Local ID		M		
Path	M	C		
Routing Path	M			
Add. Path	U	U(=)		
Port ID		M		
Result (+)			S	S(=)
AREP				M=
Receiver/Server Local ID			M=	
Service status			M	M(=)
Result (-)			S	S(=)
AREP				M=
Receiver/Server Local ID			M=	
Service status			M	M(=)

Only those argument parameters additional to the common ones are shown in Table 248 and Table 249.

Additional parameters of indication service primitives are the same as those of the corresponding request primitive.

Additional parameters of confirmation service primitives are the same as those of the corresponding response primitive.

Table 248 – Primitives issued by FAL user to FSPM

Primitive name	Source	Additional parameters	Functions
Get_attribute_all.req	FAL User	None	Conveys a request from FAL User to a target object to supply values of all attributes of the specified object class or instance
Set_attribute_all.req	FAL User	Attribute Data	Conveys a request from FAL User to a target object to write specified values of all attributes of the specified object class or instance
Get_attribute_list.req	FAL User	Attribute Count Attribute List	Conveys a request from FAL User to a target object to supply values of specified list of attributes of the specified object class or instance
Set_attribute_list.req	FAL User	Attribute Count Attribute Data	Conveys a request from FAL User to a target object to write specified values of specified list of attributes of the specified object class or instance
Get_attribute_single.req	FAL User	None	Conveys a request from FAL User to a target object to supply values of the specified attribute of the specified object class or instance

Primitive name	Source	Additional parameters	Functions
Set_attribute_single.req	FAL User	Attribute Data	Conveys a request from FAL User to a target object to write the specified values into specified attribute of the specified object class or instance
Reset.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to reset specified target object class or instance
Create.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to create an instance of the specified object class
Delete.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to delete an instance of the specified object class
Start.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to start an instance of the specified object class
Stop.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to stop an instance of the specified object class
Find_next_object_ instance.req	FAL User	Maximum Returned Values	Conveys a request from FAL User to a target device to find the identifier of the next unused instance of the specified object class
NOP.req	FAL User	None	Conveys a request from FAL User to a target object to send back a corresponding NOP response
Apply_Attributes.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to activate pending attribute values
Save.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to save its attributes contents
Restore.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to restore its attributes contents
Get_Member.req	FAL User	None	Conveys a request from FAL User to a target object to supply member(s) information from within the specified attribute
Set_Member.req	FAL User	Member Data	Conveys a request from FAL User to a target object to write member(s) information in the specified attribute
Insert_Member.req	FAL User	Member Data (optional)	Conveys a request from FAL User to a target object to inserts member(s) into the specified attribute
Remove_Member.req	FAL User	None	Conveys a request from FAL User to a target object to removes member(s) from the specified attribute
Group_Sync.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to verify that each member of a group is synchronized to System Time
Get_attribute_all.rsp	FAL User	(+) Attribute Data (-) Specific Status Code	Returns a response from FAL User to a target object to supply values of all attributes of the specified object class or instance
Set_attribute_all.rsp	FAL User	(+) None (-) Specific Status Code	Returns a response from FAL User to a target object to write specified values of all attributes of the specified object class or instance
Get_attribute_list.rsp	FAL User	(+) Attribute Count Attribute Data (-) Specific Status Code	Returns a response from FAL User to a target object to supply values of specified list of attributes of the specified object class or instance
Set_attribute_list.rsp	FAL User	(+) Attribute_count Attribute Status List (-) Specific Status Code	Returns a response from FAL User to a target object to write specified values of specified list of attributes of the specified object class or instance
Get_attribute_single.rsp	FAL User	(+) Attribute Data (-) Specific Status Code	Returns a response from FAL User to a target object to supply values of the specified attribute of the specified object class or instance
Set_attribute_single.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to write the specified values into specified attribute of the specified object class or instance
Reset.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that the instance of the specified object class has been reset

Primitive name	Source	Additional parameters	Functions
Create.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that the instance of the specified object class has been created
Delete.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that the instance of the specified object class has been deleted
Start.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that the instance of the specified object class has started
Stop.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that the instance of the specified object class has stopped
Find_next_object_ instance.rsp	FAL User	(+) Number Of List Members Instance ID List (-) Specific Status Code	Returns a response from FAL User to a target device to find the identifier of the next unused instance of the specified object class
NOP.rsp	FAL User	None	Returns a response from FAL User to a target object to confirm that the NOP has been received
Apply_Attributes.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that pending attribute values have been activated
Save.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that its attributes contents have been saved
Restore.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that its attributes contents have been restored
Get_Member.rsp	FAL User	(+) Member ID Member Data (-) Specific Status Code	Returns a response from FAL User to a target object to supply member(s) information from within the specified attribute
Set_Member.rsp	FAL User	(+) Member ID (conditional) Member Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to write member(s) information in the specified attribute
Insert_Member.rsp	FAL User	(+) Member ID Member Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to inserts member(s) into the specified attribute
Remove_Member.rsp	FAL User	(+) Member ID Member Data (-) Specific Status Code	Returns a response from FAL User to a target object to removes member(s) from the specified attribute
Group_Sync.rsp	FAL User	(+) IsSynchronized Object Specific Data (optional) (-) Specific Status Code	Conveys a request from FAL User to a target object to verify that each member of a group is synchronized to System Time

Table 249 – Primitives issued by FSPM to FAL user

Primitive name	Source	Additional parameters	Functions
Get_attribute_all.ind	FSPM	same as in .req primitive	Indicates reception of Get_attribute_all.req
Set_attribute_all.ind	FSPM	same as in .req primitive	Indicates reception of Set_attribute_all.req
Get_attribute_list.ind	FSPM	same as in .req primitive	Indicates reception of Get_attribute_list.req
Set_attribute_list.ind	FSPM	same as in .req primitive	Indicates reception of Set_attribute_list.req
Get_attribute_single.ind	FSPM	same as in .req primitive	Indicates reception of Get_attribute_single.req
Set_attribute_single.ind	FSPM	same as in .req primitive	Indicates reception of Set_attribute_single.req
Reset.ind	FSPM	same as in .req primitive	Indicates reception of Reset.req
Create.ind	FSPM	same as in .req primitive	Indicates reception of Create.req

Primitive name	Source	Additional parameters	Functions
Delete.ind	FSPM	same as in .req primitive	Indicates reception of Delete.req
Start.ind	FSPM	same as in .req primitive	Indicates reception of Start.req
Stop.ind	FSPM	same as in .req primitive	Indicates reception of Stop.req
Find_next_object_ instance.ind	FSPM	same as in .req primitive	Indicates reception of Find_next_object_instance.req
NOP.ind	FSPM	same as in .req primitive	Indicates reception of NOP.req
Apply_Attributes.ind	FSPM	same as in .req primitive	Indicates reception of Apply_Attributes.req
Save.ind	FSPM	same as in .req primitive	Indicates reception of Save.req
Restore.ind	FSPM	same as in .req primitive	Indicates reception of Restore.req
Get_Member.ind	FSPM	same as in .req primitive	Indicates reception of Get_Member.req
Set_Member.ind	FSPM	same as in .req primitive	Indicates reception of Set_Member.req
Insert_Member.ind	FSPM	same as in .req primitive	Indicates reception of Insert_Member.req
Remove_Member.ind	FSPM	same as in .req primitive	Indicates reception of Remove_Member.req
Group_Sync.ind	FSPM	same as in .req primitive	Indicates reception of Group_Sync.req
Get_attribute_all.cnf	FSPM	same as in .rsp primitive	Indicates reception of Get_attribute_all.rsp
Set_attribute_all.cnf	FSPM	same as in .rsp primitive	Indicates reception of Set_attribute_all.rsp
Get_attribute_list.cnf	FSPM	same as in .rsp primitive	Indicates reception of Get_attribute_list.rsp
Set_attribute_list.cnf	FSPM	same as in .rsp primitive	Indicates reception of Set_attribute_list.rsp
Get_attribute_signle.cnf	FSPM	same as in .rsp primitive	Indicates reception of Get_attribute_single.rsp
Set_attribute_single.cnf	FSPM	same as in .rsp primitive	Indicates reception of Set_attribute_single.rsp
Reset.cnf	FSPM	same as in .rsp primitive	Indicates reception of Reset.rsp
Create.cnf	FSPM	same as in .rsp primitive	Indicates reception of Create.rsp
Delete.cnf	FSPM	same as in .rsp primitive	Indicates reception of Delete.rsp
Start.cnf	FSPM	same as in .rsp primitive	Indicates reception of Start.rsp
Stop.cnf	FSPM	same as in .rsp primitive	Indicates reception of Stop.rsp
Find_next_object_ instance.cnf	FSPM	same as in .rsp primitive	Indicates reception of Find_next_object_instance.rsp
NOP.cnf	FSPM	same as in .rsp primitive	Indicates reception of NOP.rsp
Apply_Attributes.cnf	FSPM	same as in .rsp primitive	Indicates reception of Apply_Attributes.rsp
Save.cnf	FSPM	same as in .rsp primitive	Indicates reception of Save.rsp
Restore.cnf	FSPM	same as in .rsp primitive	Indicates reception of Restore.rsp
Get_Member.cnf	FSPM	same as in .rsp primitive	Indicates reception of Get_Member.rsp
Set_Member.cnf	FSPM	same as in .rsp primitive	Indicates reception of Set_Member.rsp
Insert_Member.cnf	FSPM	same as in .rsp primitive	Indicates reception of Insert_Member.rsp
Remove_Member.cnf	FSPM	same as in .rsp primitive	Indicates reception of Remove_Member.rsp
Group_Sync.cnf	FSPM	same as in .rsp primitive	Indicates reception of Group_Sync.rsp

8.3 Parameters of primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 250.

Table 250 – Parameters used with primitives exchanged between FAL user and FSPM

Parameter name	Description
AREP: Local identifier UCMM Record identifier Transport identifier	Identifies the entity to be used to convey the service. This parameter may use a dedicated identifier associated with a local entity, the identifier of a UCMM transaction record previously created, or the transport identifier returned by the connection establishment process and associated with the selected AR.
Receiver/Server Local ID	Generated by the Message Router ASE of the responding node. Identifies locally this invocation of the service. It is used to associate service responses with indications.
Path: Routing Path Additional path	In the request, it specifies the FAL APO or FAL APO element that is the destination of the service request. In the indication, it contains only those segments beyond the logical class segment from the service request, i.e. the optional additional information for the target APO (Add.Path).
Port ID	Indicates on which port of the device the service indication arrived.
Service status: Status Code (mandatory) Extended Status (conditional)	Provides information on the result of service execution. It is returned in all confirmed service response primitives (+ and -). Status code indicates the type of error (see IEC 61158-5-2 for details). Extended status code gives details of the status (see IEC 61158-5-2 for details).
Attribute Data	1) A stream of information containing the values of each attribute that the Class/Object defines for the request/response format. Classes/Objects which support this service shall define the format of this parameter. 2) An array of structures, predefined by the system for the given service
Attribute Count	Number of attribute numbers in the attribute list
Attribute list []	List (array) of the attribute numbers of the attributes to get from the class or object
Attribute Number	Number representing the attribute value
Attribute Status	Status information of attribute
Attribute Value	Sequence of data specific to the attribute of the object or class
Object Specific Data	Class/Instance specific parameters. Class/Instance specification shall specify the format.
Instance ID	Value assigned to identify the newly created object
Maximum Returned Values	Maximum number of Instance ID values to be returned in the response message
Number Of List Members	Number of Instance IDs specified in response message
Instance ID List	Returned Instance ID List. The Instance IDs are returned in 16 bit integer fields.
IsSynchronized	Indicates if object is synchronized to the PTP Time Master

8.4 FSPM state machines

FSPM State Machine has got only one possible state: Running.

9 Application relationship protocol machines (ARPMs)

9.1 General

This fieldbus has Application Relationship Protocol Machines for:

- connection-less application relationships,
- connection-oriented application relationships.

Connection-less relationship is of one type only.

Connection-oriented relationships fit into one of 7 transport classes:

0 Null (or Base)
1 Duplicate Detection
2 Acknowledged
3 Verified
4 Non-blocking
5 Non-blocking, Fragmenting
6 Multipoint, Fragmenting

Although similar, each of these transport classes has its own ARPM.

9.2 Connection-less ARPM (UCMM)

9.2.1 General

Functions of the connection-less ARPM are provided by the Unconnected Message Manager (UCMM) object. UCMM shall provide an unconnected request/response message services, limited to a single link that supports multiple outstanding messages. The required number of outstanding messages shall be implementation specific. The UCMM shall be present in all nodes, and shall support at least one outstanding message.

The UCMM detailed specification varies depending on the different data link layer associated which can be associated with the Type 2 application layer. The underlying data link layer defines how the UCMM is accessed and may limit the messaging which can occur across the UCMM.

9.2.2 Primitive definitions

The UCMM object shall provide the following services:

UCMM_Create	Creates an instance of transaction record. Puts UCMM into Running state. This service is local, it does not result in a PDU being sent.
UCMM_Delete	Deletes existing transaction record. Puts UCMM into Inactive state. This service is local, it does not result in a PDU being sent. This service is local, it does not result in a PDU being sent.
UCMM_Write	Writes an item of application data into Transport PDU buffer ; this results in sending the data to a specified target.
UCMM_Abort	Aborts existing transaction.

Refer to IEC 61158-5-2 for detailed definition of these services.

Primitives exchanged between UCMM and FSPM are shown in the following Table 251 and Table 252.

Table 251 – Primitives issued by FSPM to ARPM

Primitive name	Source	Associated parameters	Functions
UCMM_Create_req	FSPM	fixed tag max retries,	Conveys a request from the FSPM to the ARPM to create a transaction record.
UCMM_Delete_req	FSPM	record	Conveys a request from the FSPM to the ARPM to delete previously established transaction record.
UCMM_Write_req	FSPM	record destination ID UCMM service response timeout transaction priority application data	Conveys a request from the FSPM to the ARPM to send an item of application data using a previously created transaction record.
UCMM_Write_rsp	FSPM	record application data	Conveys a response from the FSPM, via Message Router to the ARPM to send a response to UCMM_Send_req using a previously created transaction record.
UCMM_Abort_req	FSPM	record	Aborts the transaction corresponding to the specified transaction record ; removes the packet from the local DLL if it has not yet been transmitted.

Table 252 – Primitives issued by ARPM to FSPM

Primitive name	Source	Associated parameters	Functions
UCMM_Create_cnf	ARPM	record service_status	Conveys a confirmation from the ARPM to the FSPM that transaction record has been created.
UCMM_Write_ind	ARPM	record source ID application data	Conveys an indication from the ARPM to the Message Router that data has arrived on the previously created transaction record. The Message Router then passes the indication to the appropriate application object.
UCMM_Write_cnf	ARPM	record service status application data	Conveys a confirmation from the ARPM to the Message Router of the execution of FSPM UCMM_Send_req on a previously created transaction record.

9.2.3 Parameters of primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 253.

Table 253 – Parameters used with primitives exchanged between FSPM and ARPM

Parameter name	Description
record	Identifies unambiguously the instance of the temporary connection along which the FSPM has issued a UCMM_Send request primitive. A means for such identification is not specified in this standard.
application data	Conveys FAL-User data.
fixed tag	Set to value of 0x83 for UCMM and 0x88 for Management UCMM
destination ID	Unambiguous identifier (MAC ID) of the source node of UCMM_write_req.
source ID	Unambiguous identifier (MAC ID) of the destination node of UCMM_write_req.

Parameter name	Description
service	The service parameter shall specify the delivery mechanism to use for this message and shall be one of: Request_With_Acknowledge = 2 (retries until acknowledged) Request_With_No_Response = 4 (no acknowledgement) Request_With_No_ACK = 7 (retries until response) Request_With_No_Retry = 8 (with response) NOTE These values correspond to command codes contained in the UCMM header
timeout	Duration of the transaction in microseconds. If no UCMM_Send_cnf is received before the time-out expires, the transaction is aborted and the send_status parameter shall be TIMEOUT.
retries	Conveys the maximum allowable number of retries.
create_status	Status returned with UCMM_Create_conf = - success - cannot create
send_status	Status returned with UCMM_Send_conf = - success - record_not_created - packet_too_big - no_acknowledge - aborted - timeout

9.2.4 UCMM state machines

9.2.4.1 UCMM client

9.2.4.1.1 States

The defined states and their descriptions of the UCMM Client are listed in Table 254.

Table 254 – UCMM client states

State	Description
Inactive	The record for UCMM transfer does not exist.
Running	The record for UCMM transfer has been created.
Waiting for response	Client sent an item of application data and is waiting for a response.

9.2.4.1.2 State transition diagram

Figure 33 shows the state transition diagram of UCMM client.

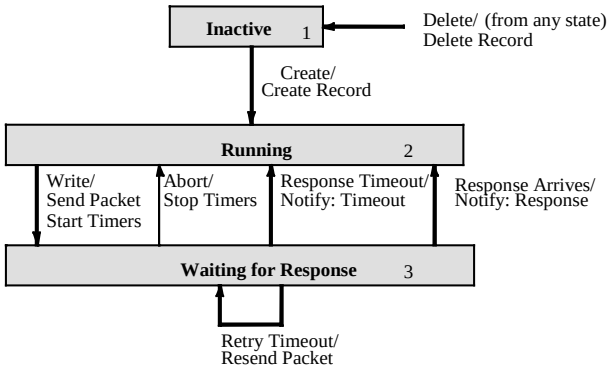


Figure 33 – State transition diagram of UCMM client9

9.2.4.1.3 State event matrix

The state transitions for the UCMM Client shall be as specified in Table 255.

Table 255 – State event matrix of UCMM client

Event	State		
	Inactive	Running	Waiting for response
Create.req	1) create record 2) transition to Running		
Delete.req	no action	1) delete record 2) increment SEQUENCE 3) transition to Inactive	1) Delete record 2) increment SEQUENCE 3) transition to Inactive
Write.req	confirm status = INVALID_RECORD	1) send packet 2) start Response Timer 3) initialize Retry Count 4) start Retry Timer 5) transition to Waiting	confirm status = BUSY
Abort.req	confirm status = INVALID_RECORD	no action	1) stop timers 2) increment SEQUENCE 3) transition to Running
Retry Timeout Request packet available			1) Decrement Retry Count IF Retry Count expired 2) Notify: Timeout-No ACK 3) Free request buffer 4) Increment Sequence # 5) Stop Response Timer 6) Transition to Running ELSE. 1) Resend packet 2) Restart Retry Timer
Response Timeout			1) confirm with status = TIMEOUT-No Response 2) increment SEQUENCE 3) transition to Running
Response Arrives		no action	1) confirm with status = SUCCESS 2) Send ACK_RESP, unless response indicates no ACK_RESP desired 3) increment SEQUENCE 4) transition to Running
ACK_REQ Arrives, sequence number matches stored value		No Action	1) Free request buffer 2) stop retry timer
ACK_REQ Arrives, sequence number not equal to stored value		No Action	No Action

The sequence number shall be set to zero at initialization. The sequence number value shall be maintained in the inactive state, to be used when the record transitions to running. The retry timeout value shall be fixed for the link at a value which guarantees that both the client and server nodes have an opportunity to transmit an unscheduled packet.

The response time-out is the response time provided when the record is created.

9.2.4.2 High-end UCMM server

9.2.4.2.1 Functions

When a packet arrives at the server an instance of a transaction shall be created if resources are available. If resources are not available the packet shall be dropped. When the instance is created, a transaction record shall be created for that instance. This record shall be active for the life of this transaction. The transaction shall end:

- when an ACK_RESP is received after a response was sent;

- when the response time timer expires; or
- when a new packet is received after a response was sent.

9.2.4.2.2 States

The defined states and their descriptions of the UCMM High-end Server are listed in Table 256.

Table 256 – High-end UCMM server states

State	Description
Inactive	The record for UCMM transfer does not exist.
Waiting for Response	Data packet arrived with new Transaction ID. New record is created. The Server is waiting for response to be sent by the Server application.
Response sent	Server application sent a response.

9.2.4.2.3 State transitions

Figure 34 shows the high-end UCMM Server state transition diagram.

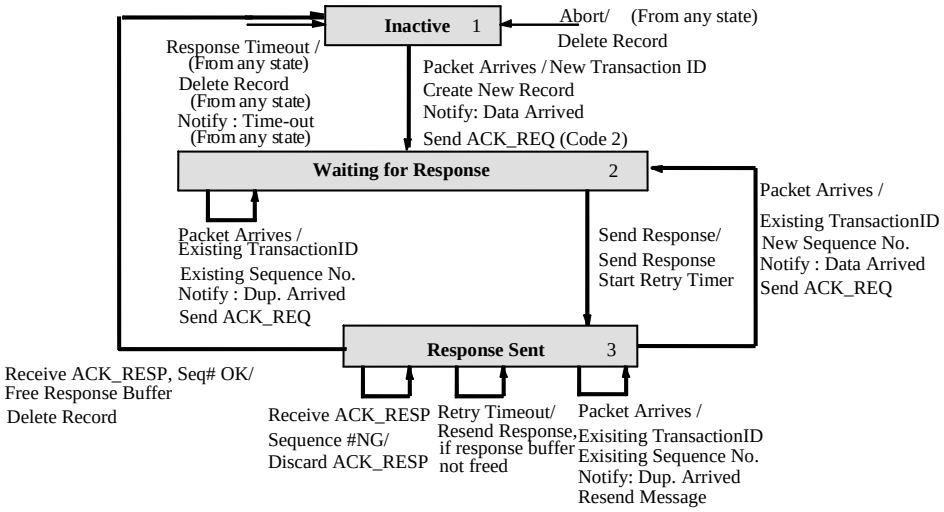


Figure 34 – State transition diagram of high–end UCMM server

9.2.4.2.4 State event matrix

The state transitions for the high-end UCMM server shall be as specified in Table 257.

The sequence number shall be set to zero at initialization. The sequence number value shall be maintained in the Inactive state, to be used when the record transitions to Running. The retry time-out value shall be fixed for the local link.

The response time-out shall be the response time provided when the record is created.

Table 257 – State event matrix of high-end UCMM server

Event	State		
	Inactive	**Waiting for response**	**Response sent**
Packet arrives request (code 2) New Transaction ID and source address	1) Create new record 2) Notify: Data Arrived 3) Send ACK_REQ 4) Start Response Timer 5) Transition to Waiting for App Response	Not Applicable	Not Applicable
Packet arrives request (code 7) New Transaction ID and source address	1) Create new record 2) Notify: Data Arrived 3) Start Response Timer 4) Transition to Waiting for App Response	Not Applicable	Not Applicable
Packet arrives Existing Transaction ID and source address New Sequence #	Not Applicable	No Action	1) Update Record 2) Notify: Data Arrived 3) Send ACK_REQ 4) Transition to Waiting for App Response
Packet arrives Existing Transaction ID Existing Sequence #	Not Applicable	1) Notify: Dup. Arrived (Optional) 2) Send ACK_REQ	1) Notify: Dup. Arrived (Optional) 2) Resend Response Message
Response Timer Expires	Not Applicable	1) Notify: Timeout 2) Delete record 3) Transition to Inactive	1) Notify: Timeout 2) Delete record 3) Transition to Inactive
Abort	Error	1) Delete record 2) Transition to Inactive	1) Delete record 2) Transition to Inactive
Send Response	Error	1) Send Response 2) Start Retry Timer 3) Transition to Response Sent	Error
Retry Timeout	Not Applicable	Not Applicable	1) Decrement Retry Count IF Retry Count expired 2) Free Response buffer 3) Stop Response Timer 4) Transition to Inactive ELSE 1) Resend Response message 2) Start Retry Timer
ACK_RESP arrives, sequence number matches stored value	No action	No action	1) Delete record 2) Transition to Inactive
ACK_RESP arrives, sequence number not equal to stored value	No action	No action	No action

The response time is the response time received in the message header.

9.2.4.3 Low-end UCMM server

9.2.4.3.1 Functions

The low-end server cannot support the following features:

- perform response message retries,
- detect duplicate messages.

9.2.4.3.2 States

The defined states and their descriptions of the UCMM Low-end Server are listed in Table 258.

Table 258 – Low-end UCMM server states

State	Description
Inactive	The record for UCMM transfer does not exist.
Waiting for Response	Data packet arrived with new Transaction ID. New record is created. The Server is waiting for response to be sent by the Server application.

9.2.4.3.3 State transitions

Figure 35 shows the low-end UCMM server state transition diagram.

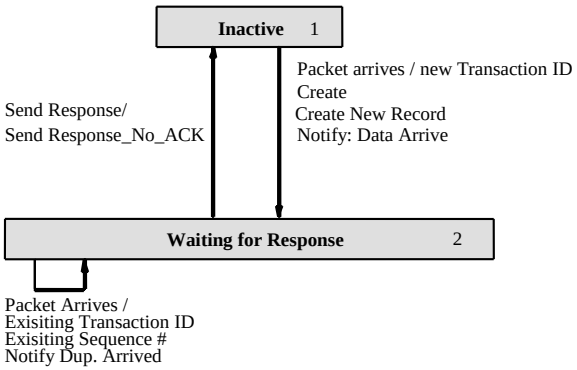


Figure 35 – State transition diagram of low–end UCMM server

9.2.4.3.4 State event matrix

The state transitions for the Low-end UCMM server shall be as specified in Table 259.

Table 259 – State event matrix of low–end UCMM server

Event	State	
	Inactive	**Waiting for response**
Packet arrives New Transaction ID and source address	1) Create new record 2) Notify: Data Arrived 3) IF Response not immediately available, Send ACK_REQ 4) Transition to Waiting for App Response	Not Applicable
Packet arrives Existing Transaction ID and source address New Sequence #	No action	No action
Packet arrives Existing Transaction ID Existing Sequence #	No action	1) Notify: Dup. Arrived (Optional)
Send Response	Error	1) Send Response_No_ACK 2) Transition to Inactive
ACK_RESP arrives	No action	No Action

9.2.5 Examples of UCMM sequences

Figure 36 shows a sequence diagram for a UCMM with one outstanding message and Figure 37 shows a sequence diagram for a UCMM with multiple outstanding messages.

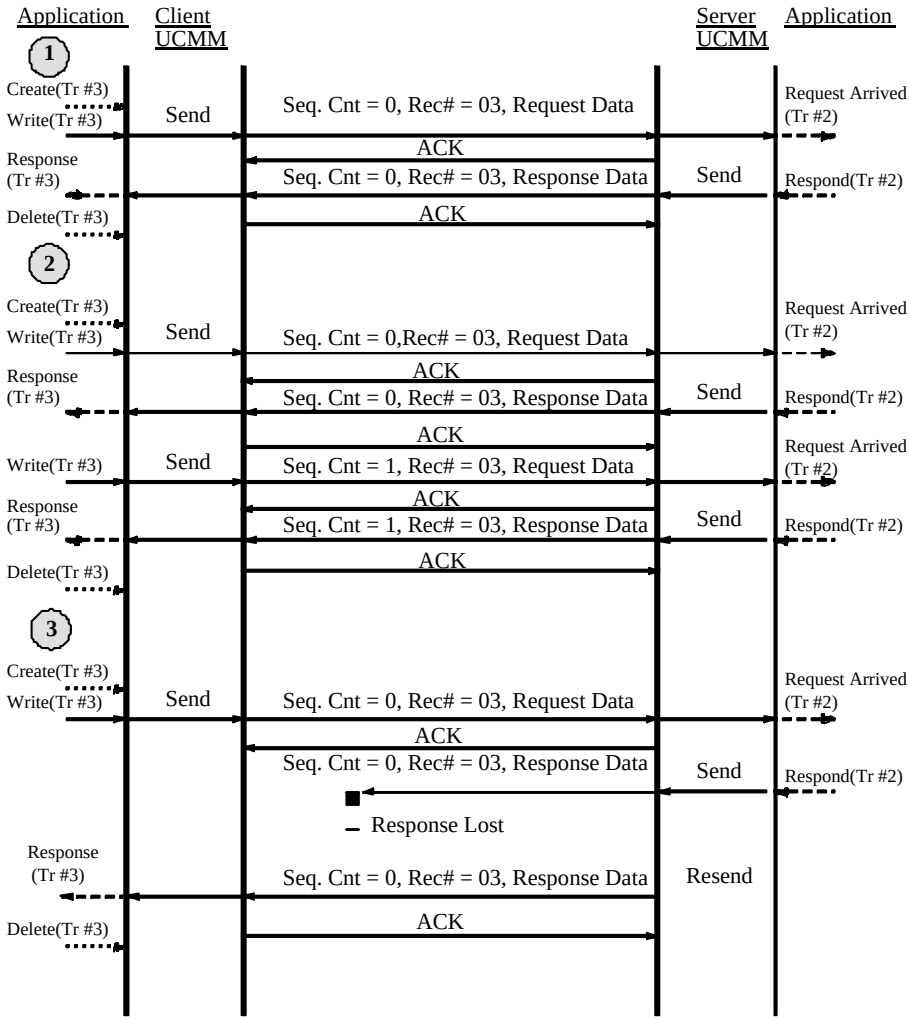


Figure 36 – Sequence diagram for a UCMM with one outstanding message

Example 1. Typical transaction: Instance is created, request is written, response is returned, and instance is deleted.

Example 2. Two transactions/same instance: In this case the instance is opened and two requests are sent.

NOTE 1 The client waits for the first response to return before issuing a second request using the same transaction number.

Example 3. Lost response: In this case the response is lost and the server issues a retry.

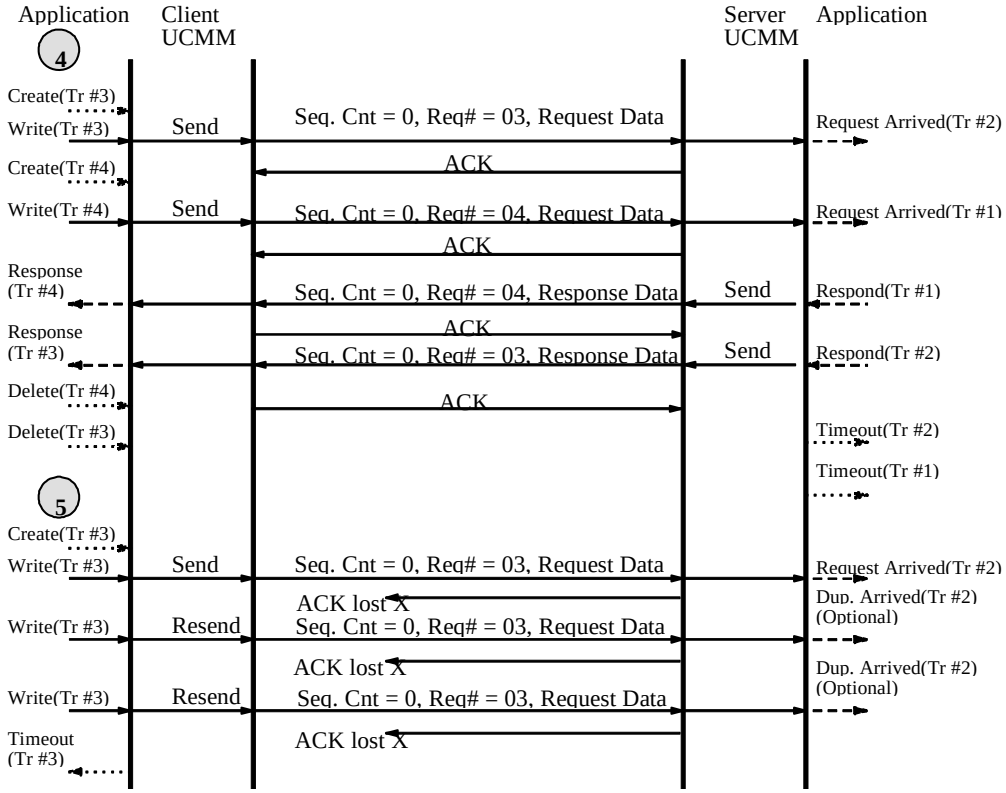


Figure 37 – Sequence diagram for a UCMM with multiple outstanding messages

Example 4. Multiple outstanding: In this case the client creates two instances and sends a second request before the first request has completed.

NOTE 2 The client uses a different transaction number for the second request.

Example 5. Timeout: In this case the server application does not respond and the client issues a timeout. The timeout causes an error status to be returned to the application. Transaction instance shall be deleted.

9.2.6 Management UCMM

Any device that has implemented the Keeper object shall also implement the Management UCMM server. With two exceptions, the Management UCMM server shall be identical to the fixed tag 0x83 UCMM server:

- transactions for the Management UCMM server shall be transmitted on fixed tag 0x88;
- the Management UCMM server shall not transmit any packets unless its node contains a Keeper object in the master state.

Any device may implement the Management UCMM client. The Management UCMM client is identical to the fixed tag 0x83 UCMM client except that transactions for the Management UCMM client shall be transmitted on fixed tag 0x88.

NOTE Messages to the Keeper object are broadcast since all devices are permitted to implement this object. To facilitate communication to the Keeper object without impacting non-Keeper devices, these messages are transmitted on a different fixed tag.

9.3 Connection-oriented ARPMs (transports)

9.3.1 Transport PDU buffer

9.3.1.1 Transport PDU format

The TPDU buffer contains a TPDU packet. The TPDU packet consists of a transport header and data. Applications shall write data to and read data from the TPDU buffers directly. Instances of transports shall write and read the transport header in the TPDU buffer while consumers shall write data to the TPDU buffer and producers shall read data from the TPDU buffer. The TPDU buffers and buffer management are not part of the communication services. This process is shown in Figure 38.

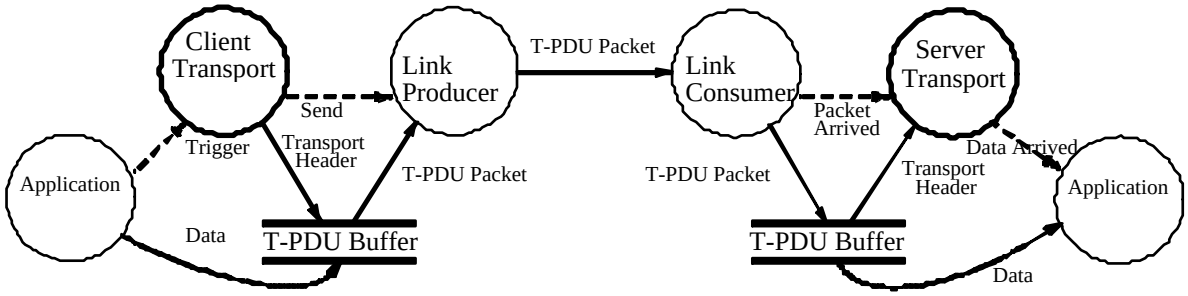


Figure 38 – TPDU buffer

9.3.1.2 Transport PDU buffer management

TPDU buffer management is implementation specific. Implementations that use single buffering, double buffering, overwrite or queuing to manage TPDU buffers are all allowed.

Implementers are responsible for ensuring buffer integrity (atomic access).

Applications are responsible for ensuring that the buffer has been initialized before a trigger is issued. Failure to initialize the buffer shall allow unknown data to be sent. This is true for producers and the client and server sides.

9.3.1.3 Notification

Transport classes can signal the application through events. The valid events are as specified in Table 260.

Table 260 – Notification

Event	Client				Server			
	Class 0	Class 1	Class 2	Class 3	Class 0	Class 1	Class 2	Class 3
Data Arrived					n	n	n	n
Duplicate Arrived						n	n	n
Acknowledgement			n					
Verification				n				

NOTE A *Duplicate Arrived* event is normally not of interest to the application; however, it could be useful in triggering a watchdog timer to indicate that the client's application is alive and triggering.

9.3.2 Transport classes

Applications interface to transport services through the supported transport classes. Table 261 lists the general–purpose transport classes that have been defined for the systems. Each transport class provides a different level of functionality. Transport class 0 provides the

minimum functionality required of a transport class, enabling data transfer between applications.

Table 261 – Transport classes

Class number	Class name
0	Null (or Base)
1	Duplicate Detection
2	Acknowledged
3	Verified
4	Non-blocking
5	Non-blocking, Fragmenting
6	Multipoint, Fragmenting

9.3.3 Common primitive definitions

Transport classes shall provide the following services:

TR_Write	Writes an item of application data into sending Transport PDU buffer, ready to be sent through the pre-established connection.
TR_Trigger	Causes an item of application data or response data previously placed in sending Transport PDU buffer, to be sent through the pre-established connection.
TR_Packet_arrived	Indicates to the Transport instance that a data packet arrived in the receiving Transport PDU buffer. This service is initiated by local action.
TR_Ack_received	Indicates to the user that a data packet arrived in the receiving Transport PDU buffer indicating an acknowledgement of arrival of previously sent data in the receiving Transport PDU buffer. This service is initiated by local action.
TR_Verify	Indicates to the user that a data packet arrived in the receiving Transport PDU buffer indicating a verification of arrival of previously sent data in the receiving user object. This service is initiated by local action.
TR_Status_Update	Indicates to the user that an updated status is present in the Transport PDU buffer.

Refer to IEC 61158-5-2 for detailed definition of these services.

Primitives exchanged between Transport instances and FSPM are shown in the following Table 262 and Table 263.

Table 262 – Primitives issued by FSPM to ARPM

Primitive name	Source	Associated parameters	Functions
TR_Write_req	FSPM	Transport identifier Application data	Conveys a request from the FSPM to the ARPM to write application data into Transport PDU buffer.
TR_Trigger.req	FSPM	Transport identifier	Conveys a request from the FSPM to the ARPM to trigger transfer of application data from Transport PDU buffer to the Link Producer.
TR_Verify.req	FSPM	Transport identifier	Conveys a request from the FSPM to the ARPM to trigger transfer of write verification data into Transport PDU buffer.

Table 263 – Primitives issued by ARPM to FSPM

Primitive name	Source	Associated parameters	Functions
TR_Packet_arrived.ind	ARPM	Transport identifier Data arrived Duplicate arrived	Conveys an indication to the FSPM that an item of data has arrived at the Link Consumer.
TR_Ack_arrived.ind	ARPM	Transport identifier	Conveys an indication to the FSPM that an acknowledgement has arrived at the Link Consumer.
TR_Status_Update.ind	ARPM	Transport identifier	Conveys an indication to the FSPM that a new status has arrived at the Link Consumer.
TR_Verify.cnf	ARPM	Transport identifier	Conveys a confirmation from the ARPM to FSPM to indicate reception of verification DLPDU.

9.3.4 Parameters of common primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 264.

Table 264 – Parameters used with primitives exchanged between FSPM and ARPM

Parameter name	Description
Transport identifier	Identifier of the instance of the transport invoked in the transaction.
Application data	Contains application data (service request/response parameters or user formatted data).
Data arrived Duplicate arrived	The Data_Arrived and Duplicate_Arrived parameters indicate whether the new packet contains new data, or whether it is just a duplicate from a previously received packet.

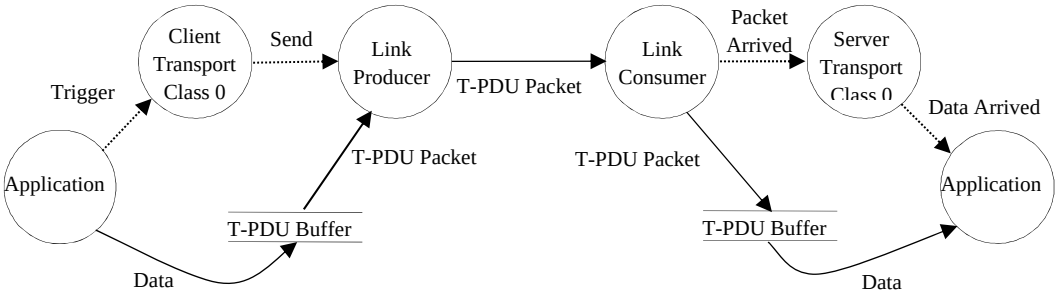
9.3.5 Transport state machines – class 0

9.3.5.1 Functions

Transport class 0 is the simplest transport class, and can use either a point–to–point or multipoint connection. Class 0 is typically used to transmit or receive inputs on a multipoint connection or outputs on a point–to–point connection.

Possible uses of transport class 0 include sampled inputs, standard outputs, cyclic block transfers, diagnostic events, and communication between controllers and operator interface devices.

Since transport class 0 does not detect duplicate data packets, the target application is responsible for detecting them. A duplicate data packet is a retransmission of the last data packet sent. Figure 39 shows the actions which take place during a data transfer using client transport class 0 and server transport class 0.



NOTE The client transport instance does not write a sequence count to the client-side T-PDU buffer, and the server transport instance does not read a sequence count from the server-side T-PDU buffer. The transport header is defined as a don't care

Figure 39 – Data flow diagram using a client transport class 0 and server transport class 0

Figure 40 shows the sequence in which actions take place when transferring data using client transport class 0 and server transport class 0.

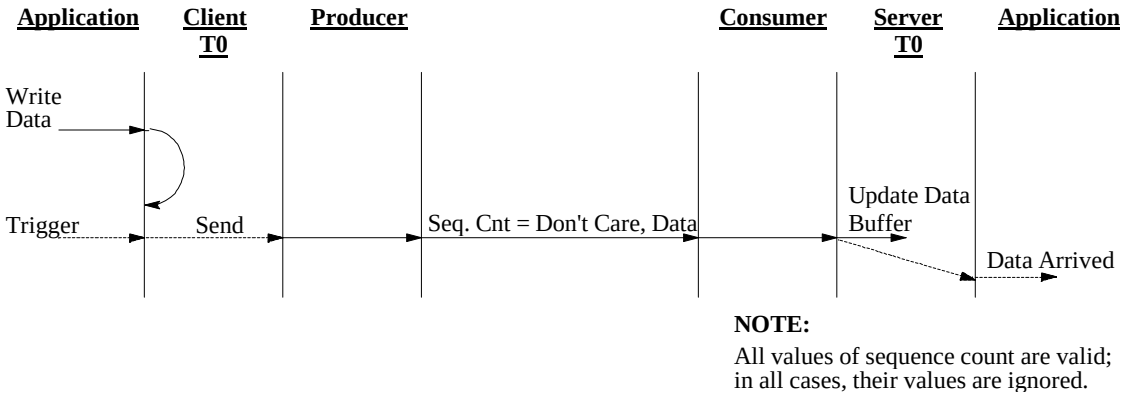


Figure 40 – Sequence diagram of data transfer using transport class 0

9.3.5.2 Transport class 0 client

9.3.5.2.1 Functions

The function of client transport class 0 is just to pass the **trigger** service from the application to the producer as a **send** service. It is useful for the client transport class 0 to be shown from a high-level design view. In actual implementations, the application may trigger the producer directly. An idle state is provided for consistency with other transport classes. In the **idle** state all events, except **delete** and **start**, are ignored.

A class 0 client transport is responsible for:

- accepting the client application's trigger that it has written a data packet to the client–side TPDU buffer,
- writing a transport header to the TPDU buffer for each packet that the client application writes to the client–side TPDU buffer (optional),
- triggering the producing node of the network connection to produce the TPDU packet on the link and send it to the consuming node of the network connection.

9.3.5.2.2 States

The defined states and their descriptions of the Class 0 Transport Client are listed in Table 265.

Table 265 – Class 0 transport client states

State	Description
Non-existent	The instance of Transport Class 0 does not exist.
Idle	The instance of Transport Class 0 has been created but not yet started.
Running	The instance of Transport Class 0 has been created and started.

9.3.5.2.3 State transitions

State transition diagram of the class 0 client transport is shown in Figure 41.

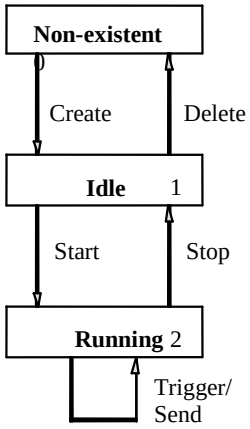


Figure 41 – Class 0 client STD

9.3.5.2.4 State event matrix

State event matrix of the class 0 client transport is shown in Table 266.

Table 266 – Class 0 client SEM

Event	State		
	Non-existent	**Idle**	**Running**
Create	Transition to **Idle**	Error	Error
Delete	Error	Transition to **Non-existent**	Error
Start	Error	Transition to **Running**	Error
Stop	Error	No Action	Transition to **Idle**
Trigger	Error	No Action	Send

9.3.5.3 Transport class 0 server

9.3.5.3.1 Function

The function of server transport class 0 is to pass the **packet arrived** event from the consumer to the producer as a **data arrived** event. It is useful for the server transport class 0 to be shown from a high-level design view. In actual implementations, the application may receive the event directly from the consumer. In the **idle** state all packet arrivals are ignored.

A class 0 server transport is responsible for:

- accepting the notification from the consuming node of the network connection that it has written a data packet to the server–side TPDU buffer,
- locating and reading the transport header of each packet that the consuming node of the network connection writes to the server–side TPDU buffer,
- notifying the server application that data has been written to the server–side TPDU buffer.

9.3.5.3.2 States

The defined states and their descriptions of the Class 0 Transport Server are listed in Table 267.

Table 267 – Class 0 transport server states

State	Description
Non-existent	The instance of Transport Class 0 does not exist.
Idle	The instance of Transport Class 0 has been created but not yet started.
Running	The instance of Transport Class 0 has been created and started.

9.3.5.3.3 State transitions

State transitions of the class 0 server transport are shown in Figure 42.

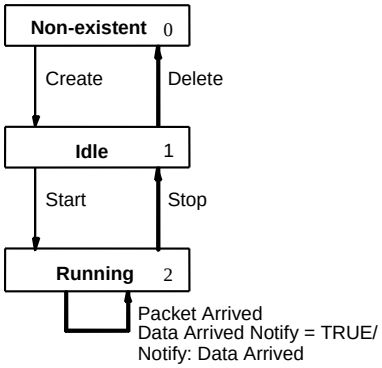


Figure 42 – Class 0 server STD

9.3.5.3.4 State event matrix

State event matrix of the class 0 server transport is shown in Table 268.

Table 268 – Class 0 server SEM

Event	State		
	Non-existent	**Idle**	**Running**
Create	Transition to **Idle**	Error	Error
Delete	Error	Transition to **Non-existent**	Error
Start	Error	Transition to **Running**	Error
Stop	Error	No action	Transition to **Idle**
Packet Arrived & Data Arrived Notify = TRUE	No action	No action	Notify: Data Arrived

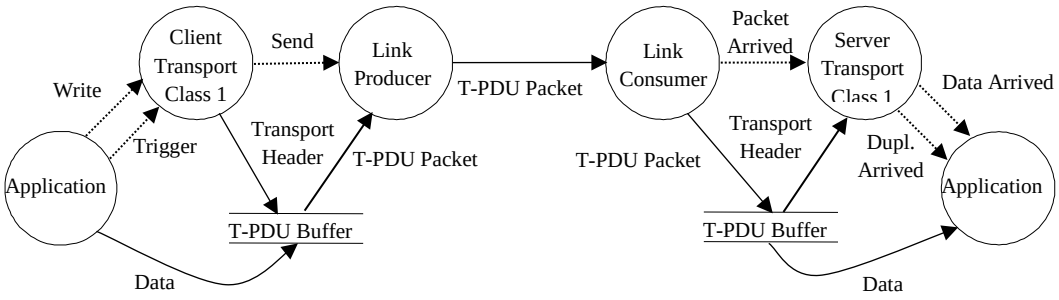
9.3.6 Transport state machines – class 1

9.3.6.1 Functions

Transport class 1, like class 0, shall allow either a point–to–point or multipoint connection. Unlike class 0, which has no transport header, the class 1 transport shall include a sequence number, which is used to detect the delivery of duplicate data packets.

NOTE 1 Class 1 is preferred to class 0 for targets because the target application does not have to perform duplicate detection, which reduces the overhead required in those end nodes that support the change of state transport trigger. Class 1 allows an efficient change of state trigger, since was designed to allow change of state data transfers.

Figure 43 shows the actions which take place during data transfer using a class 1 client transport instance and a class 1 server transport instance.



NOTE The T-PDU packet comprises the transport header from the client transport and data from the application.

Figure 43 – Data flow diagram using client transport class 1 and server transport class 1

Figure 44 shows the sequence in which actions take place when transferring data using client transport class 1 and server transport class 1.

NOTE 2 The sequence count is incremented with every write. Also, in the case where the packet is lost on a new data sample, and the packet is later triggered and sent, the data received by the client is treated as new data. This is because this is the first time the server has received this sequence count. This mechanism provides fault tolerance for those samples that change infrequently.

NOTE 3 It is an implementation choice to decide whether to update the data buffer upon receipt of duplicate data or not, based on the potential impact of the additional processing.

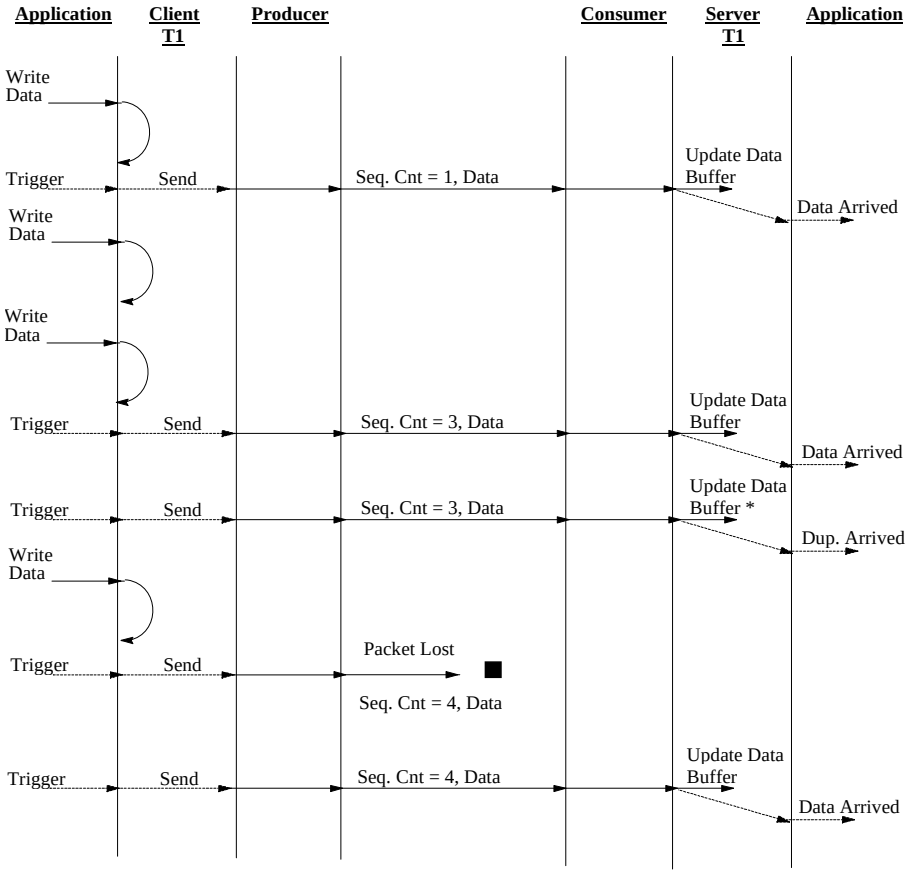


Figure 44 – Sequence diagram of data transfer using client transport class 1 and server transport class 1

Transport class 1 can be used in conjunction with a heartbeat timer to keep a connection open when there are periods when no data is transmitted. The heartbeat timer shall be initialized every time the producing node of the network connection transmits a packet, and shall be set so that its maximum value, or the maximum time between data transmissions, is less than the path time–out value for the network connection. When the heartbeat timer reaches its maximum value, it triggers the client transport instance so that the producing node produces and retransmits the packet in the client–side TPDU buffer before the connection times out. The consuming node of the network connection, recognizing that the packet has the same sequence count as the last packet it received, may or may not overwrite the packet in the server–side TPDU buffer; that implementation decision is left to the product developers, since they can best assess the impact of additional processing.

Applications could use this transport class to distinguish between new samples and old samples that were sent to maintain the connection. To maintain the connection, a timer may be used to trigger the production of the current data in the TPDU buffer. To transmit new data the application would first write to the TPDU buffer and then trigger the client transport.

Applications could use this transport class to distinguish between new samples and old samples that were sent to maintain the connection. To maintain the connection, the data arrived and duplicated arrived events may be ORed together to reset a timer. If this timer were to expire, the application would know that no data was received within the designated time. The data arrived event would identify new samples of data. This may allow applications to reduce their overhead by only processing new data samples.

Possible uses of transport class 1 include sampled inputs, standard outputs, cyclic block transfers, diagnostic events, and communication between controllers and operator interface devices.

9.3.6.2 Transport class 1 client

9.3.6.2.1 Functions

Transport class 1 starts with the behaviour in class 0 and adds a sequence count. This sequence count is incremented by the client transport class when data is written to the data buffer. When the transport receives a **write** event, it shall increment the sequence count. When a **trigger** event is received the transport shall update the sequence count in the TPDU buffer and invoke the **send** service.

Applications could use this transport class to distinguish between new samples and old samples that were sent to maintain the connection. To maintain the connection, a timer may be used to trigger the production of the current data in the TPDU buffer. To transmit new data the application would first write to the TPDU buffer and then trigger the client transport.

A class 1 client transport is responsible for:

- accepting the client application's trigger that it has written a data packet to the client–side TPDU buffer,
- writing a transport header to the TPDU buffer for each packet that the client application writes to the client–side TPDU buffer,
- initializing the sequence count in the transport header (for the first packet transmitted) or incrementing the sequence count (for subsequent packets of the same transmission),
- triggering the producing node of the network connection to produce the TPDU packet on the link and send it to the consuming node of the network connection.

9.3.6.2.2 States

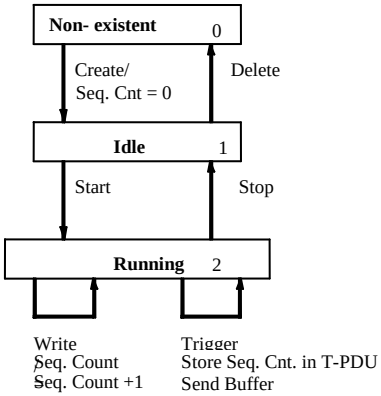
The defined states and their descriptions of the Class 1 Transport Client are listed in Table 269.

Table 269 – Class 1 transport client states

State	Description
Non-existent	The instance of Transport Class 1 does not exist.
Idle	The instance of Transport Class 1 has been created but not yet started.
Running	The instance of Transport Class 1 has been created and started.

9.3.6.2.3 State transitions

State transitions of the class 1 client transport are shown in Figure 45.



NOTE The sequence count is set to 0 by the create service. It is then incremented after every *write* event until it reaches a maximum value of 2^{16}-1. After that it rolls over to 0.

Figure 45 – Class 1 client STD

9.3.6.2.4 State event matrix

State transitions of the class 1 client transport are shown in Table 270.

Table 270 – Class 1 client SEM

Event	State		
	Non-existent	**Idle**	**Running**
Create	1) Transition to **Idle** 2) Set seq. count = 0	Error	Error
Delete	Error	Transition to **Non-existent**	Error
Start	Error	Transition to **Running**	Error
Stop	Error	No Action	Transition to **Idle**
Write	Error	No Action	1) Seq. Cnt = Seq. Cnt + 1
Trigger	Error	No Action	1) Store Seq. Cnt. in TPDU Buffer 2) Send

9.3.6.3 Transport class 1 server

9.3.6.3.1 Functions

Transport class 1 starts with the behaviour in class 0 and adds a sequence count. The received sequence count is compared with the previous sequence count. If they are equal, the received packet is considered to be a duplicate packet. If they are not equal, the received data is considered to be a new sample. There is no attempt to count how many sequence counts have changed between samples.

Applications could use this transport class to distinguish between new samples and old samples that were sent to maintain the connection. To maintain the connection, the data arrived and duplicated arrived events may be ORed together to reset a timer. If this timer were to expire, the application would know that no data was received within the designated time. The data arrived event would identify new samples of data. This may allow applications to reduce their overhead by only processing new data samples.

A class 1 server transport is responsible for:

- accepting the notification from the consuming node of the network connection that it has written a data packet to the server–side TPDU buffer,

- locating and reading the transport header of each packet that the consuming node of the network connection writes to the server–side TPDU buffer,
- triggering the server application that data has been written to the server–side TPDU buffer,
- comparing the sequence count of each packet with that of the previous packet,
- notifying the server application that the most recently arrived packet is a duplicate of the previous one when their sequence counts match.

9.3.6.3.2 States

The defined states and their descriptions of the Class 1 Transport Server are listed in Table 271.

Table 271 – Class 1 transport server states

State	Description
Non-existent	The instance of Transport Class 1 does not exist.
Idle	The instance of Transport Class 1 has been created but not yet started.
Ready to run	The instance of Transport Class 1 has been created and started, waiting for a first data packet to arrive.
Running	The instance of Transport Class 1 has been created and started, first data packet has arrived.

9.3.6.3.3 State transitions

State transitions of the class 1 server transport are shown in Figure 46.

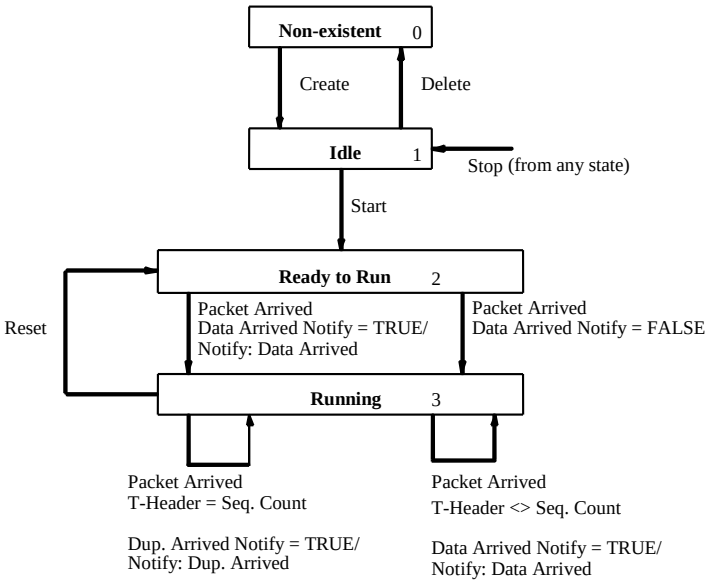


Figure 46 – Class 1 server STD

9.3.6.3.4 State event matrix

State transitions of the class 1 server transport are shown in Table 272.

Table 272 – Class 1 server SEM

Event	State/			
	Non-existent	**Idle**	**Ready to run**	**Running**
Create	Transition to **Idle**	Error	Error	Error
Delete	Error	Transition to **Non-existent**	Error	Error
Start	Error	Transition to **Ready to Run**	Error	Error
Stop	Error	No action	Transition to **Idle**	Transition to **Idle**
Reset	Error	Error	No Action	Transition to **Ready to Run**
Packet Arrived Transport Header <> Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = TRUE	No action	No action	Transition to **Running** Notify: Data Arrived	Notify: Data Arrived
Packet Arrived Transport Header = Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = TRUE	No action	No action	Transition to **Running** Notify: Data Arrived	Notify: Duplicate Arrived
Packet Arrived Transport Header <> Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = FALSE	No action	No action	Transition to **Running** Notify: Data Arrived	Notify: Data Arrived
Packet Arrived Transport Header = Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = FALSE	No action	No action	Transition to **Running** Notify: Data Arrived	No action
Packet Arrived Transport Header <> Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = TRUE	No action	No action	Transition to **Running**	No action
Packet Arrived Transport Header = Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = TRUE	No action	No action	Transition to **Running**	Notify: Duplicate Arrived
Packet Arrived Transport Header <> Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = FALSE	No action	No action	Transition to **Running**	No action
Packet Arrived Transport Header = Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = FALSE	No action	No action	Transition to **Running**	No action

9.3.7 Transport state machines – class 2

9.3.7.1 Functions

Transport class 2 starts with the behaviour in class 1 and adds a return connection that acknowledges the delivery of a packet. The producer connection from the client node can be point to point or multipoint. The connection from the server to the client is point to point.

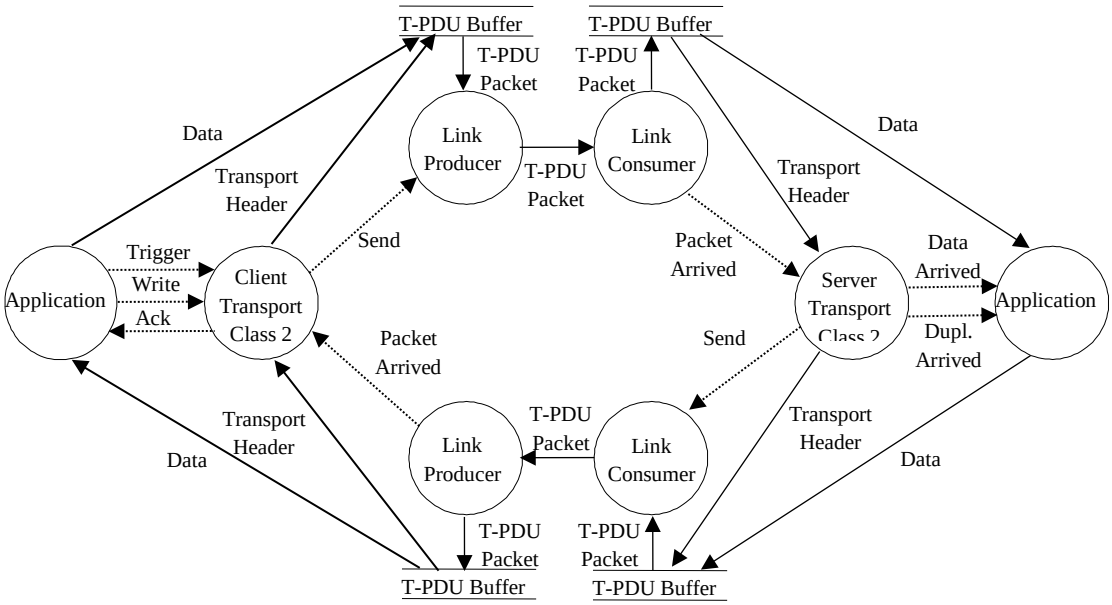
This transport class allows the client application to be notified that the server has received the transmitted packet. This acknowledgement tells the client application only that the data arrived. This does not mean that the server application has read the buffer or that a new sample can be sent without overwriting the buffer.

The main advantage of this class over class 3 (Verified) is that the acknowledgement is returned immediately after the packet arrives at the consumer, while the verify class requires the application to initiate the return verification. The delays associated with receiving the acknowledgement are small, approximately twice the time that is required for a one way transmission.

Transport class 2 uses two network connections: the originator–to–target connection, which can be either point–to–point or multipoint; and the return connection, which shall be point–to–point. A multipoint connection using transport class 2 shall have a point–to–point return connection corresponding to each client–to–server connection. The acknowledgement notifies the client transport instance that the consuming node of the network connection has received the packet.

If a server transport instance does not acknowledge receipt of the packet, the client transport instance triggers the client application to retransmit the most recently sent packet; if the client–to–server connection is one of the network connections of a multipoint connection, data is resent over all of the network connections of the multipoint connection. The server application can return data to the client application along with its acknowledgement.

Figure 47 shows the actions which take place during a data transfer using client transport class 2 and server transport class 2



NOTE 1 The client and server transports each use two buffers: one for transmitting, and one for receiving.

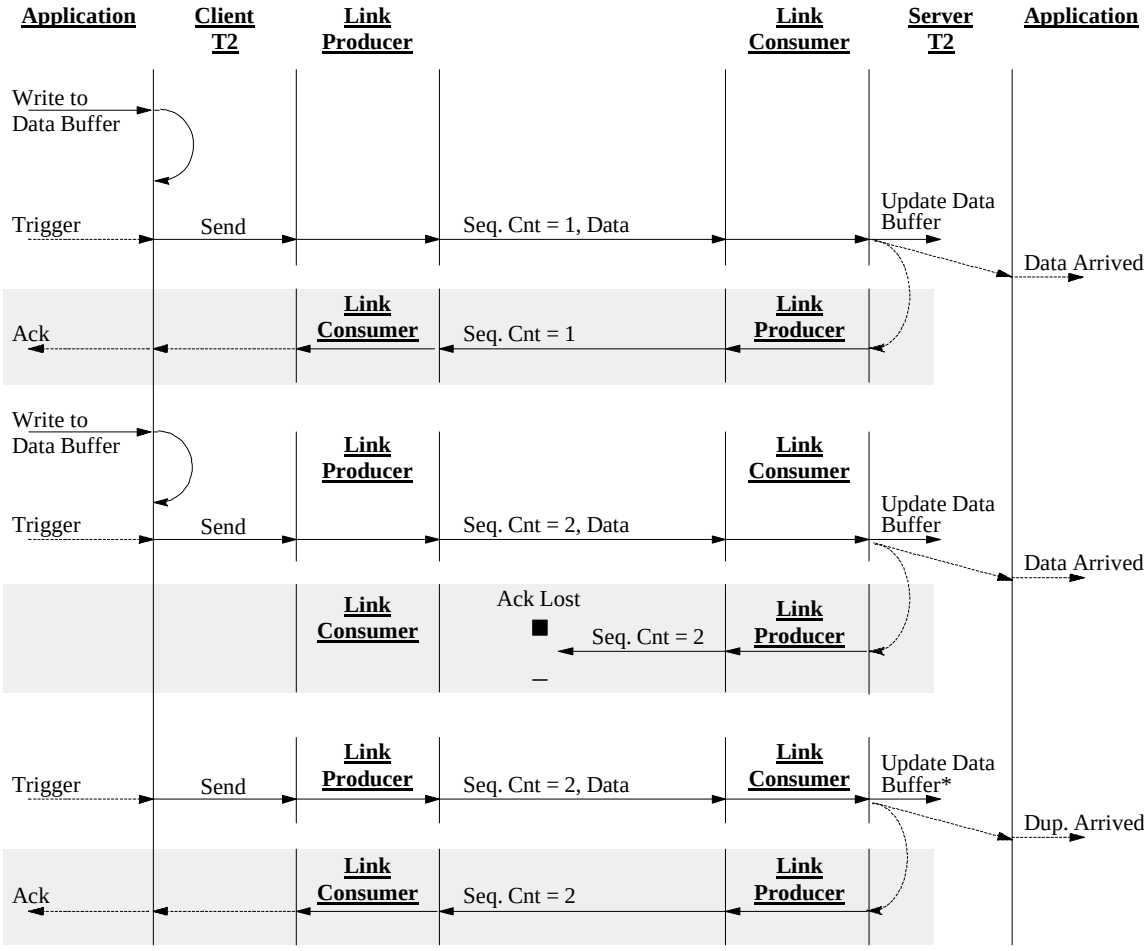
NOTE 2 The target application is not involved in sending the acknowledgement.

Figure 47 – Data flow diagram using client transport class 2 and server transport class 2

The server transport instance initiates and sends the acknowledgement to the client transport instance to notify the client application that it can write the next data packet to the client–side TPDU transmit buffer. Each acknowledgement shall be received by the client transport instance before the next packet is sent over the same client–to–server network connection(s).

Figure 48 shows the sequence of actions which take place during data transfer using client transport class 2 and server transport class 2. In the example depicted, the acknowledgement

is the only data transmitted in the server–to–client direction. The unshaded areas in Figure 48 indicate client–to–server data transmission, and the shaded areas indicate server–to–client transmission.



* Implementors must decide whether to update the data buffer upon receipt of duplicate data packets, since they can best assess the impact of the additional processing.

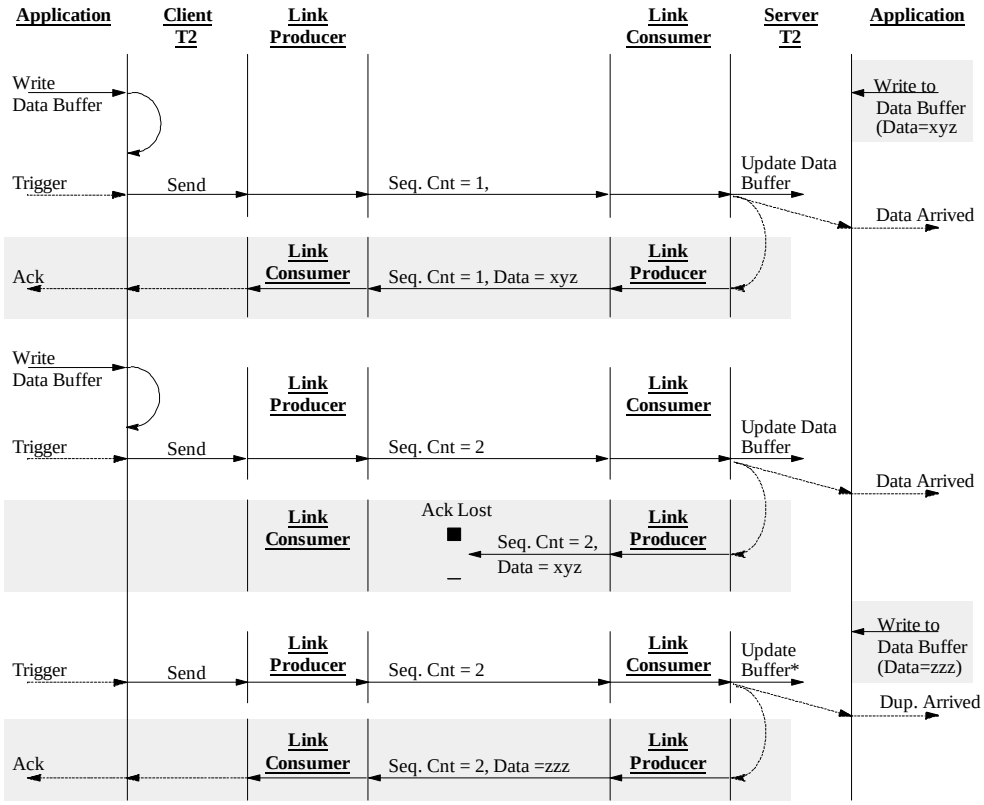
Figure 48 – Diagram of data transfer using client transport class 2 and server transport class 2 without returned data

At times, the server–to–client transmission comprises the acknowledgement and additional data.

Figure 49 shows the sequence in which actions take place during data transfer using client transport class 2 and server transport class 2 when the acknowledgement and data are returned to the client.

NOTE The returned data is written asynchronously to the receipt of data from the client.

The unshaded areas in Figure 49 indicate client–to–server data transmission, and the shaded areas indicate server–to–client data transmission.



* Implementors must decide whether to update the data buffer upon receipt of duplicate data packets, since they can best assess the impact of the additional processing.

Figure 49 – Sequence diagram of data transfer using client transport class 2 and server transport class 2 with returned data

Possible uses of transport class 2 include master/slave applications and communication between controllers and operator interface devices.

9.3.7.2 Transport class 2 client

9.3.7.2.1 Functions

In the client–to–server direction, a class 2 client transport is responsible for:

- accepting the client application's trigger that it has written a data packet to the client–side TPDU transmit buffer;
- writing a transport header to the client–side TPDU transmit buffer for each packet that the client application writes to it; the transport header for each packet shall include a sequence count;
- initializing the sequence count in the transport header (for the first packet transmitted) or incrementing the sequence count (for subsequent packets of the same transmission);
- triggering the producing node of the network connection to produce the TPDU packet on the link and send it to the consuming node of the network connection.

In the server–to–client direction, a class 2 client transport is responsible for:

- accepting notification from the consuming node of the return network connection that it has written data (the server transport instance's acknowledgement) to the client–side TPDU receive buffer;
- locating and reading the transport header of each data packet that the consuming node of the return network connection writes to the client–side TPDU receive buffer;

- notifying the client application that the consuming node of the return network connection has written data to the client–side TPDU receive buffer.

The application can update the data buffer or retrigger data before all of the acknowledgements have been received. If the client transport is retriggered while waiting for an acknowledgement, the previous acknowledgement received register is cleared. This means that the client transport class shall wait for all active consumers to return an acknowledgement before notifying the application.

9.3.7.2.2 States

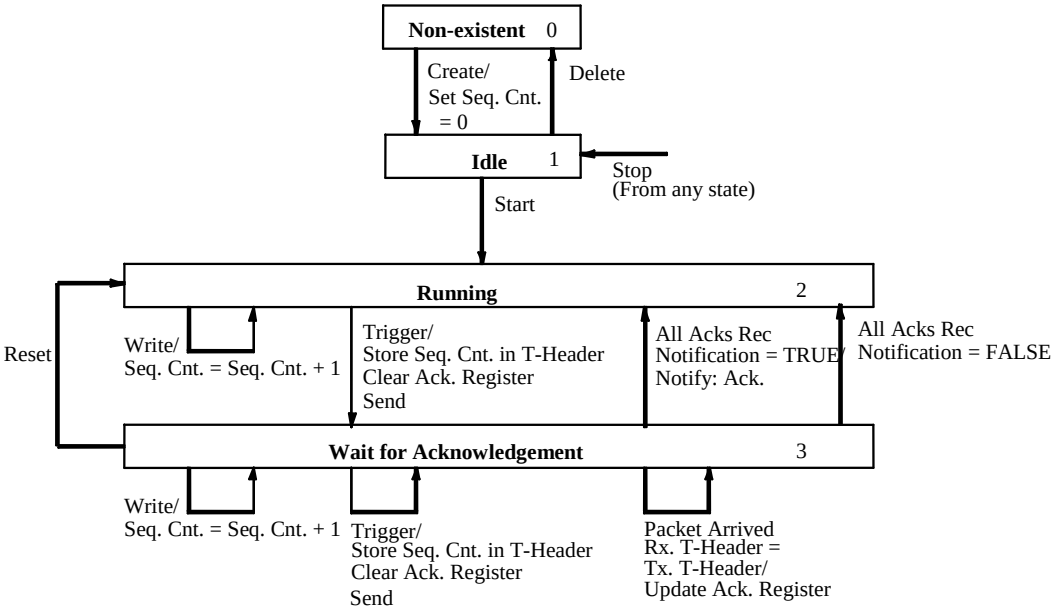
The defined states and their descriptions of the Class 2 Transport Client are listed in Table 273.

Table 273 – Class 2 transport client states

State	Description
Non-existent	The instance of Transport Class 2 does not exist.
Idle	The instance of Transport Class 2 has been created but not yet started.
Running	The instance of Transport Class 2 has been created and started.
Wait for acknowledgement	The instance of Transport Class 2 has been created and started, acknowledgement is pending.

9.3.7.2.3 State transitions

State transitions of the class 2 client transport are shown in Figure 50.



NOTE The sequence count is set to 0 by the create service. It is then incremented after every *write* event until it reaches a maximum value of 2^{16}-1. After that it rolls over to 0.

Figure 50 – Class 2 client STD

9.3.7.2.4 State event matrix

State transitions of the class 2 client transport are shown in Table 274.

Table 274 – Class 2 client SEM

Event	State			
	Non-existent	**Idle**	**Running**	**Wait for ack**
Create	1) Seq. Cnt. = 0 2) Transition to **Idle**	Error	Error	Error
Delete	Error	Transition to **Non-existent**	Error	Error
Start	Error	Transition to **Running**	Error	Error
Stop	Error	No Action	Transition to **Idle**	Transition to **Idle**
Reset	Error	Error	No Action	Transition to **Running**
Write	Error	No Action	1) Seq. Cnt. = Seq. Cnt. + 1 2) Write TPDU	1) Seq.Cnt. = Seq.Cnt. +1 2) Write TPDU
Trigger	Error	No Action	1) Store Seq. Cnt. in TPDU 2) Clear Ack. register 3) Send 4) Transition to **Wait for Ack.**	1) Store Seq. Cnt. in TPDU 2) Clear Ack. Register 3) Send
Packet Arrived Rx. T-Header = Tx. T-Header	Error	No Action	No Action	1) Update Ack. Register
All Ack Received Ack. Notify = TRUE	Error	No Action	Error	1) Notify: Ack. 2) Transition to **Running**
All Ack Received Ack. Notify = FALSE	Error	No Action	Error	1) Transition to **Running**

NOTE The STD and SEM for client classes 2 and 3 are nearly identical. Verify has been substituted for acknowledgement.

9.3.7.3 Transport class 2 server

9.3.7.3.1 Functions

Server transport class 2 starts with the behaviour from transport class 1 and adds a return connection from the server to the client that is used to acknowledge a message. This return connection is used to identify the acknowledgement message.

After receiving a **packet arrived** event, the server transport shall read the received transport header, store it in the transmit transport header and invoke the **send** service. The application is not involved in the acknowledgement. The producer connection from the client node can be point to point or multipoint. The connection from the server to the client is point to point.

In the client–to–server direction, a class 2 server transport is responsible for:

- accepting the notification from the consuming node of the network connection that it has written a data packet to the server–side TPDU receive buffer;
- locating and reading the transport header of each packet that the consuming node of the network connection writes to the server–side TPDU receive buffer;
- comparing the sequence count of each packet in the server–side TPDU receive buffer with that of the previous packet;
- notifying the server application that the most recently arrived packet is a duplicate of the previous one when their sequence counts match;
- notifying the server application that data has been written to the server–side TPDU receive buffer. (ACK may contain server data).

In the server–to–client direction, a class 2 server transport is responsible for:

- writing the transport header (including sequence count) of the data in the server–side TPDU receive buffer to the server–side TPDU transmit buffer;
- triggering the producing node of the return network connection to produce the TPDU packet on the link and send it to the consuming node of the return network connection.

9.3.7.3.2 States

The defined states and their descriptions of the Class 2 Transport Server are listed in Table 275.

Table 275 – Class 2 transport server states

State	Description
Non-existent	The instance of Transport Class 2 does not exist.
Idle	The instance of Transport Class 2 has been created but not yet started.
Running	The instance of Transport Class 2 has been created and started.

9.3.7.3.3 State transitions

State transitions of the class 2 server transport are shown in Figure 51.

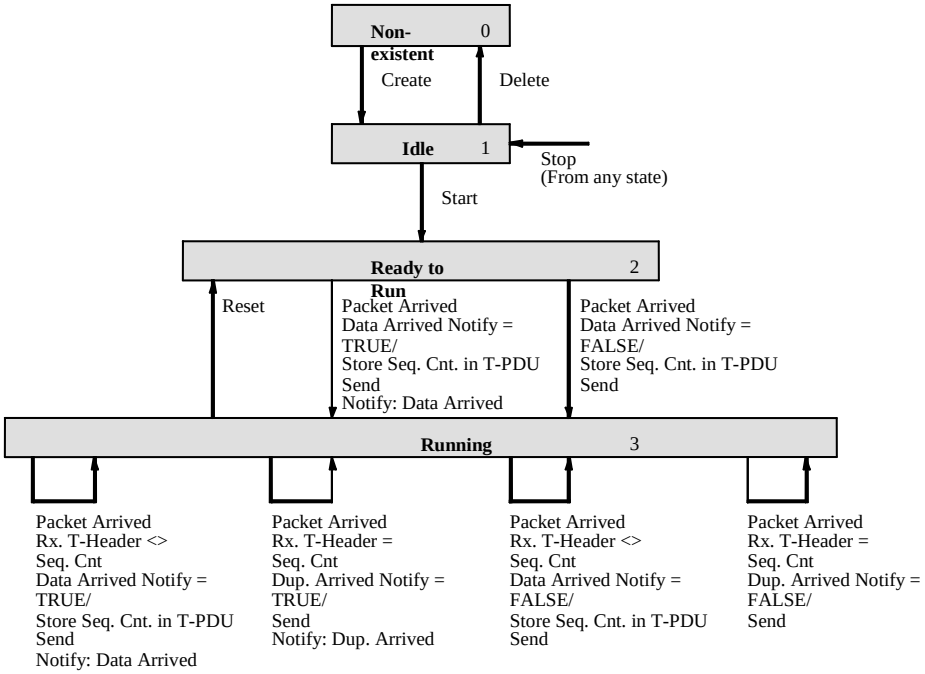


Figure 51 – Class 2 server STD

The send service is invoked for sequence counts that match and sequence counts that differ. This behaviour is required to ensure that a node that has sent an acknowledgement that was lost shall retransmit the acknowledgement on a retry.

9.3.7.3.4 State event matrix

State transitions of the class 2 server transport are shown in Table 276.

Table 276 – Class 2 server SEM

Event	State			
	Non-existent	**Idle**	**Ready to run**	**Running**
Create	Transition to **Idle**	Error	Error	Error
Delete	Error	Transition to **Non-existent**	Error	Error
Start	Error	Transition to **Ready to Run**	Error	Error
Stop	Error	No action	Transition to **Idle**	Transition to **Idle**
Reset	Error	Error	No action	Transition to **Ready to Run**
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = TRUE Duplicate Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3)Notify: Data Arrived 4) Transition to **Running**	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3)Notify: Data Arrived
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3)Notify: Data Arrived 4) Transition to **Running**	1) Send (Implies Ack) 2) Notify: Duplicate Arrived
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = TRUE Duplicate Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3)Notify: Data Arrived 4) Transition to **Running**	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3)Notify: Data Arrived
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3) Notify: Data Arrived 4) Transition to **Running**	1) Send (Implies Ack)
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = FALSE Duplicate Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3) Transition to **Running**	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack)
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3) Transition to **Running**	1) Send (Implies Ack) 2) Notify: Duplicate Arrived
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = FALSE Duplicate Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3) Transition to **Running**	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack)
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3) Transition to **Running**	1) Send (Implies Ack)

9.3.8 Transport state machines – class 3

9.3.8.1 Functions

Transport class 3 starts with the behaviour in class 1 and adds a return connection that verifies that the application has received the packet. The sequence count that shall be returned with the verify is the same as the sequence count sent with the data. This return connection is used to identify the verification message.

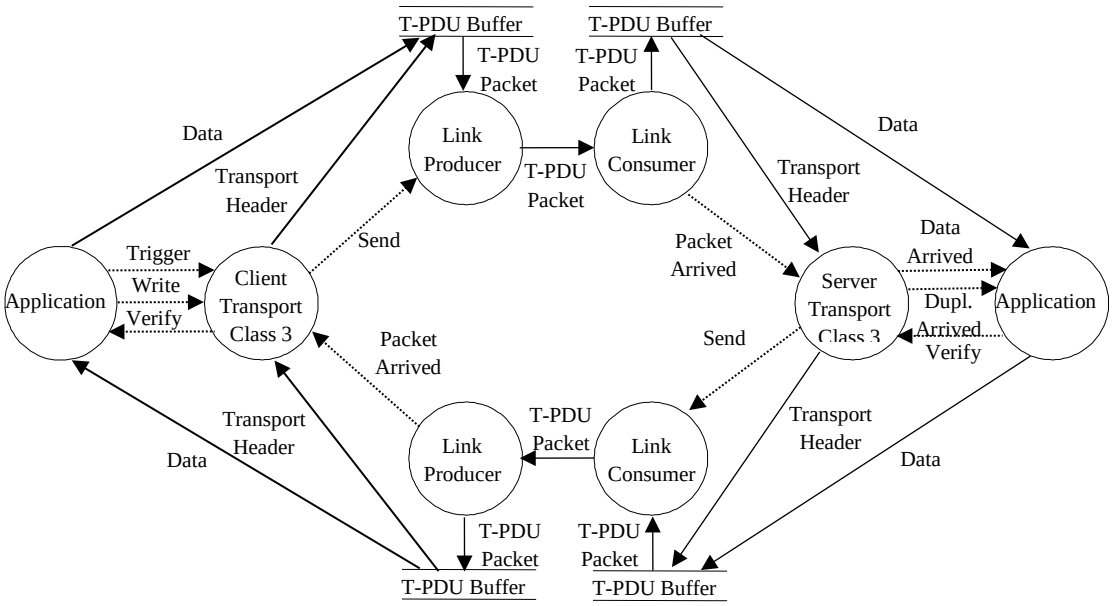
This transport class allows the application to be notified that the server application has responded to the transmitted packet. This verification tells the client application not only that the data arrived but that the server application has read the buffer and that a new sample can be sent without overwriting the buffer.

The main advantage of this class over class 2 (Acknowledgement) is that verification conveys an application response. The disadvantage is that the delay in receiving the verification can be very difficult to predict.

Transport class 3 uses two connections: the originator–to–target connection, which can be either point–to–point or multipoint; and the return connection, which shall be point–to–point. A multipoint connection using transport class 3 shall have a point–to–point return connection corresponding to each client–to–server connection. The verification notifies the client application that the server application has received and read the transmitted data. If a server application does not verify receipt of the packet, the client application retransmits the most recently sent packet; if the client–to–server connection is a multipoint connection, the data is retransmitted over all of the network connections of the multipoint connection.

The server application initiates and sends the verification after the consuming node of the network connection has written the packet to the TPDU buffer, after the transport has read the packet's transport header, and after the application has read the packet. The server's verification signals the client application that the server–side TPDU receive buffer is ready to accept the next data packet.

Figure 52 shows the actions which take place during a data transfer using client transport class 3 and server transport class 3.

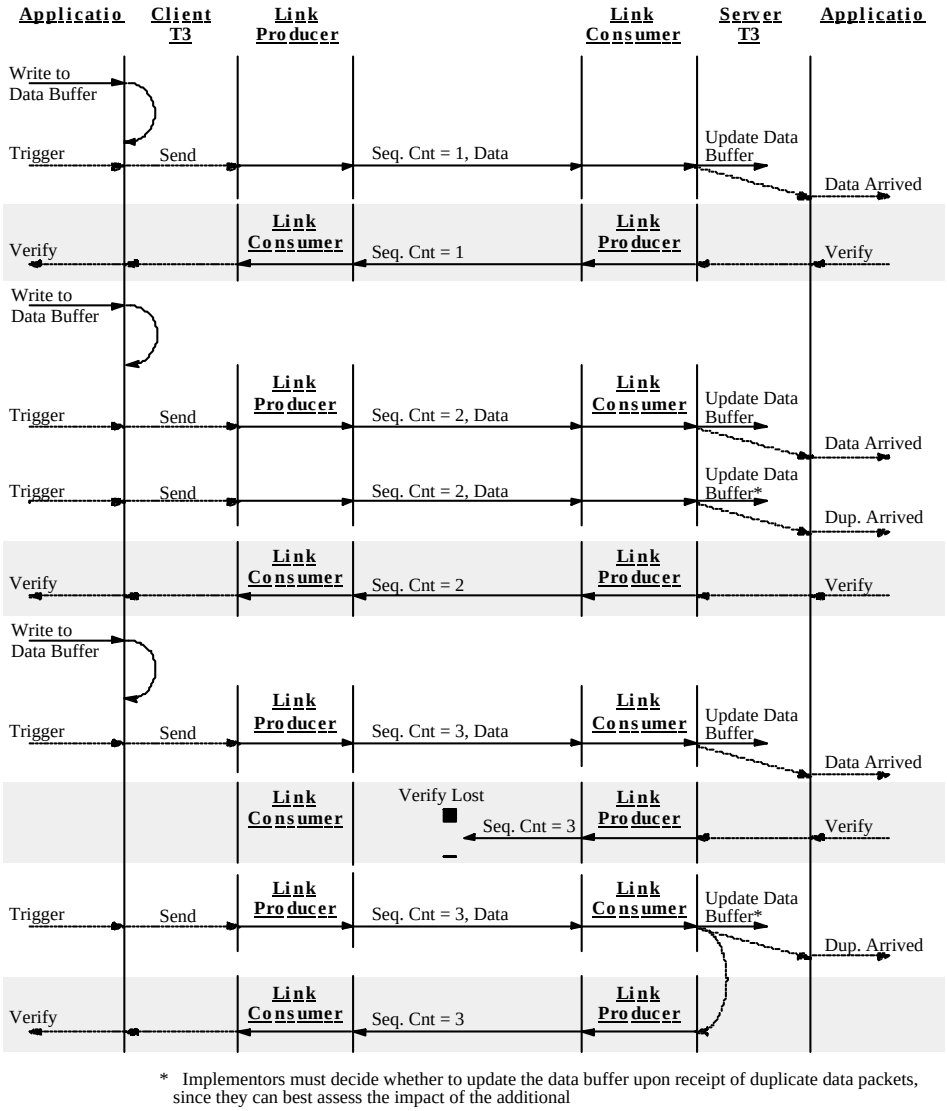


NOTE 1 The client and server transports each use two buffers: one for transmitting, and one for receiving.

NOTE 2 The server application can return data to the client application using the connection established for the verification.

Figure 52 – Data flow diagram using client transport class 3 and server transport class 3

Figure 53 shows the sequence of actions which take place during data transfer using client transport class 3 and server transport class 3. In the example depicted, the verification is the only data transmitted in the server–to–client direction. The unshaded areas in Figure 53 indicate client–to–server data transmission, and the shaded areas indicate server–to–client transmission.



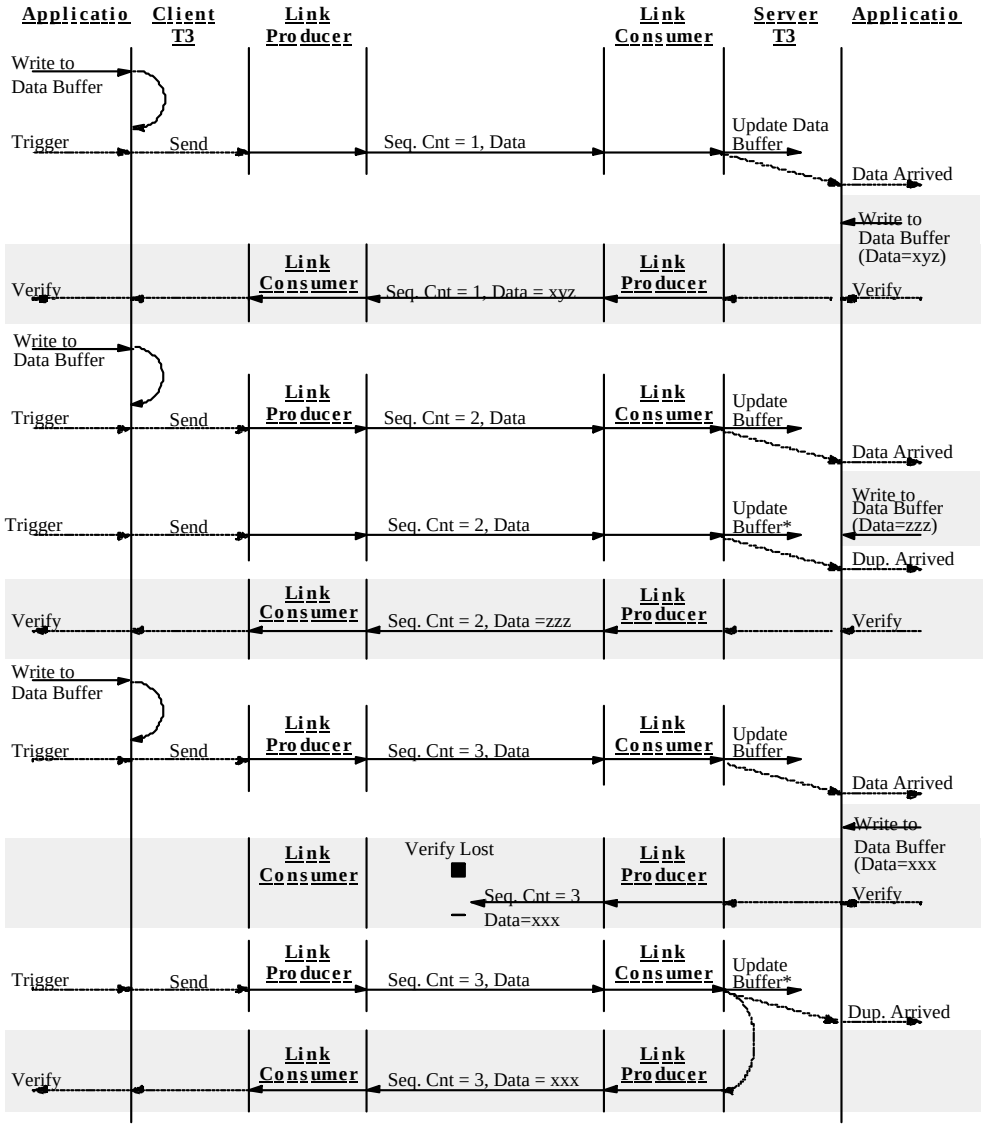
* Implementors must decide whether to update the data buffer upon receipt of duplicate data packets, since they can best assess the impact of the additional

Figure 53 – Sequence diagram of data transfer using client transport class 3 and server transport class 3 without returned data

At times, the server–to–client transmission comprises the verification and additional data. Figure 54 shows the sequence in which actions take place during data transfer using client transport class 3 and server transport class 3 when the verification and data are returned to the client.

NOTE The returned data is written asynchronously to the receipt of data from the client.

The unshaded areas in Figure 54 indicate client–to–server data transmission, and the shaded areas indicate server–to–client transmission.



* Implementors must decide whether to update the data buffer upon receipt of duplicate data since they can best assess the impact of the additional processing.

Figure 54 – Sequence diagram of data transfer using client transport class 3 and server transport class 3 with returned data

Possible uses of transport class 3 include change–of–state inputs and outputs, asynchronous [independent] block transfers, event acknowledgements, communication between controllers and operator interface devices, uploads, downloads, and messaging.

9.3.8.2 Transport class 3 client

9.3.8.2.1 Functions

In the client–to–server direction, a class 3 client transport is responsible for:

- accepting the client application's trigger that it has written a data packet to the client–side TPDU transmit buffer;
- writing a transport header to the client–side TPDU transmit buffer for each packet that the client application writes to it;
- initializing the sequence count in the transport header (for the first packet transmitted) or incrementing the sequence count (for subsequent packets of the same transmission);

- triggering the producing node of the network connection to produce the TPDU packet on the link and send it to the consuming node of the network connection.

In the server–to–client direction, a class 3 client transport is responsible for:

- accepting notification from the consuming node of the return network connection that it has written data (the server transport instance's verification) to the client–side TPDU receive buffer;
- locating and reading the transport header of each data packet that the consuming node of the return network connection writes to the client–side TPDU receive buffer;
- notifying the client application that the consuming node of the return network connection has written data to the client–side TPDU receive buffer.

The application can update the data buffer or retrigger data before all of the verifications have been received. If the client transport is retriggered while waiting for verification, the previous verification received register is cleared. This means that the client transport class shall wait for all active consumers to return verification before notifying the application.

9.3.8.2.2 States

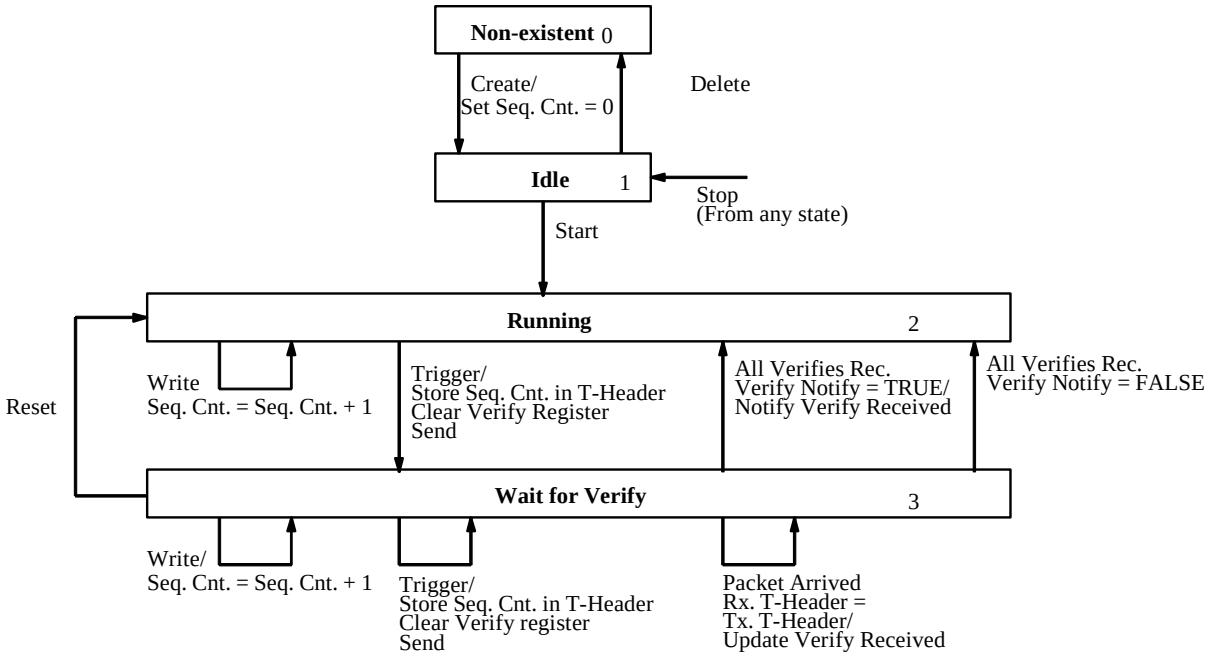
The defined states and their descriptions of the Class 3 Transport Client are listed in Table 277.

Table 277 – Class 3 transport client states

State	Description
Non-existent	The instance of Transport Class 3 does not exist.
Idle	The instance of Transport Class 3 has been created but not yet started.
Running	The instance of Transport Class 3 has been created and started.
Wait to verify	The instance of Transport Class 3 has been created and started, verification is pending.

9.3.8.2.3 State transitions

State transitions of the class 3 client transport are shown in Figure 55.



NOTE The sequence count is set to 0 by the create service. It is then incremented after every *write* event until it reaches a maximum value of 2^{16}-1. After that it rolls over to 0.

Figure 55 – Class 3 client STD

9.3.8.2.4 State event matrix

State transitions of the class 3 client transport are shown in Table 278.

Table 278 – Class 3 client SEM

Event	State			
	Non-existent	**Idle**	**Running**	**Wait for verify**
Create	1) Seq. Cnt. = 0 2) Transition to **Idle**	Error	Error	Error
Delete	Error	Transition to **Non-existent**	Error	Error
Start	Error	Transition to **Running**	Error	Error
Stop	Error	No action	Transition to **Idle**	Transition to **Idle**
Reset	Error	Error	No Action	Transition to **Running**
Write	Error	No action	Seq. Cnt. = Seq. Cnt. + 1	Seq. Cnt. = Seq. Cnt. + 1
Trigger	Error	No action	1) Store Seq. Cnt. in TPDU 2) Clear Verify register 3) Send 4) Transition to **Wait for Verify**	1) Store Seq. Cnt. in TPDU 2) Clear Verify register 3) Send
Packet Arrived Rx. T- header = Tx. T-Header	No action	No action	No Action	1) Update Verify Register
All Verifies Received Notification = TRUE	No action	No action	Error	1) Notify: Verify 2) Transition to **Running**
All Verifies Received Notification = FALSE	No action	No action	Error	1) Transition to **Running**

NOTE The STD and SEM for client classes 2 and 3 are nearly identical. Acknowledgement has been substituted for verification.

9.3.8.3 Transport class 3 server

9.3.8.3.1 Functions

Transport class 3 starts with the behaviour in class 1 and adds a return connection that verifies that the application has received the packet. It is the responsibility of the application to verify receipt of data. The sequence count that shall be returned with the verification is the same as the sequence count sent with the data.

The application is required to read the TPDU buffer before invoking the verify service.

In the client–to–server direction, a class 3 server transport is responsible for:

- accepting the notification from the consuming node of the network connection that it has written a data packet to the server–side TPDU receive buffer;
- locating and reading the transport header of each packet that the consuming node of the network connection writes to the server–side TPDU receive buffer;
- comparing the sequence count of each packet in the server–side TPDU receive buffer with that of the previous packet;
- notifying the server application that the most recently arrived packet is a duplicate of the previous one when their sequence counts match;
- notifying the server application that data has been written to the server–side TPDU receive buffer.

In the server–to–client direction, a class 3 server transport is responsible for:

- writing the transport header of the data in the server–side TPDU receive buffer to the server–side TPDU transmit buffer;
- correlating that transport header with any data that the server application wants to send to the client application and has written to the server–side TPDU transmit buffer;
- triggering the producing node of the return network connection to produce the TPDU packet on the link and send it to the consuming node of the return network connection.

NOTE Data from the application can be return from the server to the client along the verification connection.

The send service is invoked when verify is received from an application or when a duplicate transmission is received while in the running state. Retransmitting when a previously verified packet has been received allows this class to recover from lost verifications.

9.3.8.3.2 States

The defined states and their descriptions of the Class 3 Transport Server are listed in Table 279.

Table 279 – Class 3 transport server states

State	Description
Non-existent	The instance of Transport Class 3 does not exist.
Idle	The instance of Transport Class 3 has been created but not yet started.
Read to run	The instance of Transport Class 3 has been created and started, waiting for the first packet to arrive.
Wait for verify	The instance of Transport Class 3 has been created and started, the first packet arrived, waiting for verification to be requested by the application.
Running	The instance of Transport Class 3 has been created and started, verification has been processed.

9.3.8.3.3 State transitions

State transitions of the class 3 server transport are shown in Figure 56.

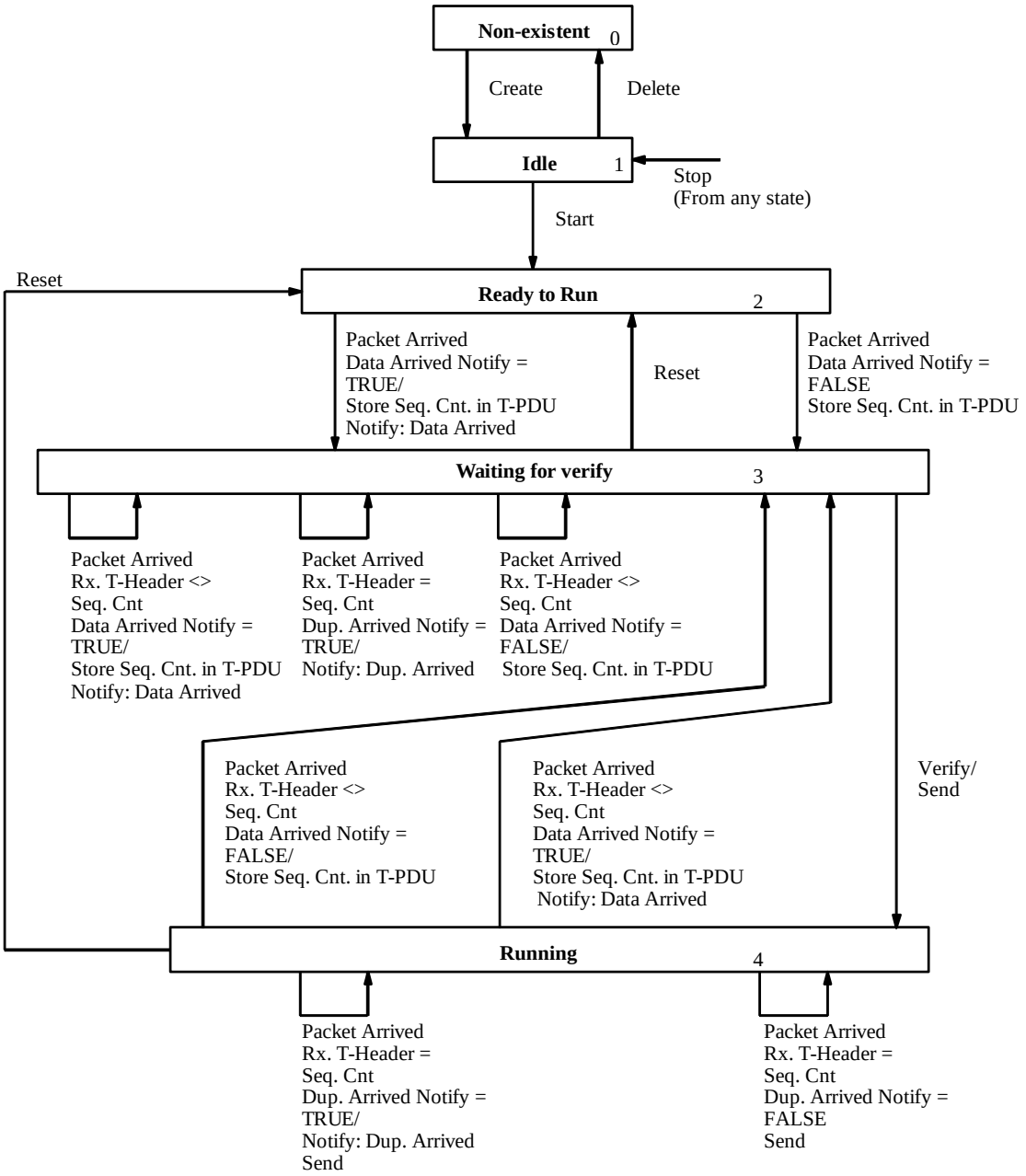


Figure 56 – Class 3 server STD

9.3.8.3.4 State event matrix

State transitions of the class 3 server transport are shown in Table 280.

Table 280 – Class 3 server SEM

Event	State				
	Non-existent	Idle	Ready to run	Waiting for verify	Running
Create	Transition to Idle	Error	Error	Error	Error
Delete	Error	Transition to Non-existent	Error	Error	Error
Start	Error	Transition to Ready to Run	Error	Error	Error
Stop	Error	No action	Transition to Idle	Transition to Idle	Transition to Idle
Reset	Error	Error	No action	Transition to Ready to Run	Transition to Ready to Run
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = TRUE Dup Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify
Packet Arrived Rx. T-Header = Seq.Count Data Arrived Notify = TRUE Dup Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify	1) Notify: Dup. Arrived	1) Notify: Dup. Arrived 2) Send (Implies Verify)
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = TRUE Dup Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify
Packet Arrived Rx. T-Header = Seq.Count Data Arrived Notify = TRUE Dup Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify	No action	1) Send (Implies Verify)
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = FALSE Dup Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Transition to Waiting for Verify	1) Store Seq. Cnt. in TPDU	1) Store Seq. Cnt. in TPDU 2) Transition to Waiting for Verify
Packet Arrived Rx. T-Header = Seq.Count Data Arrived Notify = FALSE Dup Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Transition to Waiting for Verify	1) Notify: Dup. Arrived	1) Notify: Dup. Arrived 2) Send (Implies Verify)
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = FALSE Dup Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Transition to Waiting for Verify	1) Store Seq. Cnt. in TPDU	1) Store Seq. Cnt. in TPDU 2) Transition to Waiting for Verify
Packet Arrived Rx. T-Header = Seq.Count Data Arrived Notify = FALSE Dup Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Transition to Waiting for Verify	No action	1) Send (Implies Verify)
Verify	Error	No action	No action	1) Send (Implies Verify) 2) Transition to Running	No action

9.3.9 Transport state machines – classes 4, 5, 6

NOTE Subclause contents from the previous edition have been obsoleted.

9.3.10 Transport state machines – class 4

NOTE Subclause contents from the previous edition have been obsoleted.

9.3.11 Transport state machines – class 5

NOTE Subclause contents from the previous edition have been obsoleted.

9.3.12 Transport state machines – class 6

NOTE Subclause contents from the previous edition have been obsoleted.

10 DLL mapping protocol machine 1 (DMPM 1)

10.1 General

Clause 10 defines the requirements for mapping the transmission of TPDU's over a Type 2 data-link layer.

The DLL Mapping Protocol Machine is common to all the AREP types.

The network connections that interface to the Data-link layer include the link producer(s) and link consumer(s).

Data flow of the link producer and consumer on an instance of Transport connection is shown in Figure 57.

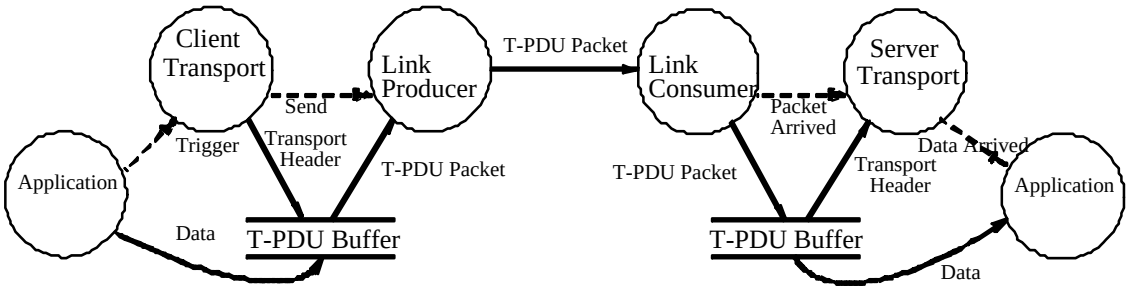


Figure 57 – Data flow diagram for a link producer and consumer

10.2 Link producer

The link producer shall be responsible for fetching data from the TPDU buffer and producing it on the data-link after the **send** event has been received. TPDU buffer management schemes such as overwrite, double buffered, triple buffered are not excluded, but since they are implementation specific, they are not specified here.

The link producer shall read the TPDU packet from the TPDU buffer and transmit it unchanged. While the TPDU buffer is shown as a single buffer, it does not imply that the transport header and the data are stored in consecutive locations. An implementation that stored the transport header and data in non-consecutive locations is permitted as long as the link producer can find all parts of the TPDU packet.

The application, not the link producer, shall be responsible for ensuring that the TPDU buffer has been initialized before the first trigger has been issued.

10.3 Link consumer

The link consumer is responsible for receiving TPDU packets, storing them in the TPDU buffer and notifying the server transport class that a packet has arrived. Data buffer management schemes such as overwrite, double buffered, triple buffered are not excluded, but since they are implementation specific, they are not specified here.

The link consumer shall write the TPDU packet unchanged to the TPDU buffer. While the TPDU buffer is shown as a single buffer, it does not imply that the transport header and the data are stored in consecutive locations. There are no restrictions on actual implementations. An implementation that stored the transport header and data in non-consecutive locations is permitted as long as the consumer can find both parts of the TPDU packet.

10.4 Primitive definitions

10.4.1 Primitives exchanged between DMPM and ARPM

List of primitives exchanged between DMPM and ARPM is shown in Table 281 and Table 282.

Table 281 – Primitives issued by ARPM to DMPM

Primitive names	Source	Associated parameters	Functions
Send.req	ARPM	Service Dest-id TPDU	This primitive is used to request the DMPM to transfer the TPDU to the Data-link. It passes the TPDU to the DMPM as a DLSDU.

Table 282 – Primitives issued by DMPM to ARPM

Primitive names	Source	Associated parameters	Functions
Packet_Arrived.ind	DMPM	Service Source-id TPDU	This primitive is used to pass the TPDU received as a data-link layer service data unit to a designated ARPM.

10.4.2 Parameters of ARPM/DMPM primitives

The parameters used with the primitives exchanged between the ARPM and the DMPM are described in Table 283.

Table 283 – Parameters used with primitives exchanged between ARPM and DMPM

Parameter name	Description
TPDU	This parameter carries the transport PDU containing the user data and Transport header.
DL-SDU	This parameter carries the received Data-link SDU.

10.4.3 Primitives exchanged between data-link layer and DMPM

The request and confirmation services used to queue Lpackets with the data-link layer shall be of the form:

NOTE The primitives shown in Table 284 and their parameters are defined in detail in IEC 61158-3-2.

Table 284 – Primitives exchanged between data-link layer and DMPM

Primitive names	Source	Associated parameters
DL_generic_Tag.req	ARPM	Request_DLS_user_id (handle) user_data [] QoS (priority) generic_TAG
DL_generic_Tag.cnf	Data-link layer	Request_DLS_user_id (handle) TX_status
DL_generic_Tag.ind	Data-link layer	Request_DLS_user_data [] generic_TAG
DL_fixed_Tag.req	ARPM	Request_DLS_user_id (handle) user_data [] QoS (priority) fixed_TAG destination-DLE-ID
DL_fixed_Tag.cnf	Data-link layer	Request_DLS_user_id (handle) TX_status
DL_fixed_Tag.ind	Data-link layer	user_data [] fixed_TAG source-DLE-ID
DL_tone.ind	Data-link layer	DLS_cycle

10.4.4 Parameters of DMPM/Data-link Layer primitives

The parameters used with the primitives exchanged between the DMPM and the data-link layer are identical to those defined in the data-link layer Service Definition (IEC 61158-3-2).

The parameters are described in Table 285.

Table 285 – Parameters used with primitives exchanged between DMPM and Data-link

Parameter name	Description
Request_DLS_user_id	Provides a local means of pairing the resulting confirm primitive with the causative request primitive.
User_data	Data to be transmitted between DLS-users without alteration by the DLS-provider.
QoS	Selects one of the following priorities: urgent scheduled high low
Generic_TAG	Connection identification or service point address identifying the remote DLSAP(s) to which the DLS is to be provided.
Fixed_TAG	Specifies the destination service point in the station identified by the DLS- Destination-DLE-ID address.
Destination-DLE-ID	Address identifying the destination station for the message using its MAC ID address.
Source-DLE-ID	Address identifying the local station from which the fixed tag DLSDU has been sent. It is a station MAC ID address on the local link.
TX_status	Status of the requested transmission: a) “OK” message successfully sent; b) “TxAbort” sending process failed; c) “Flushed” message has been removed from the pending queue before sending.
DLS_cycle	Interval count for the Network Update Time (NUT) which has just been received within the overall cycle of scheduled access intervals.

10.4.5 Network connection ID

The Network Connection ID shall be a 32-bit value built as follows:

- Bits 31 to 24 shall be reserved and shall be set to zero;
- Bits 23 to 16 shall contain a node MAC ID depending on selected connection type;
- Bits 15 to 0 shall contain the 16-bit connection number.

Table 286 shows options for selection of Connection ID.

Table 286 – Selection of connection ID

Connection type	MAC ID	Connection number
Multipoint	MAC of Producer	Unique by Device
Point to Point	MAC of Consumer	Unique by Device

The generic tag for the Lpackets exchanged over a connection and described in IEC 61158-4-2, shall be built as follows, based on the contents of the CID fields:

- the first transmitted octet of the generic tag shall contain the MAC ID;
- the second transmitted octet of the generic tag shall contain the least significant octet of the connection number;
- the third transmitted octet of the generic tag shall contain the most significant octet of the connection number.

To guarantee that connection IDs are unique on a link the following rules shall be used when choosing CIDs:

- the producer shall choose the CID for a multipoint transport;
- the consumer shall choose the CID for a point-to-point transport;
- on a CP 2/1 link, the node responsible for the choice of the CID (either producer or consumer) shall insert its MAC ID into bits 23 to16 of the CID in the Forward_Open;
- a multipoint CID shall not be reused until all connections associated with the CID have been closed or timed out;
- receiving an I'm alive fixed tag packet from a node shall immediately close any connections with that device.

10.5 DMPM state machine

10.5.1 DMPM states

10.5.1.1 Link producer

10.5.1.1.1 Link producer states

The defined states and their descriptions of the Link Producer are listed in Table 287 below:

Table 287 – Link producer states

State	Description
Non-existent	The Link Producer does not exist.
Running	The Link Producer has been created and is running.

10.5.1.1.2 State transition diagram

Figure 58 shows the state transition diagram of a link producer.

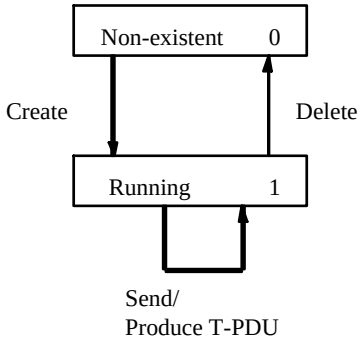


Figure 58 – State transition diagram for a link producer

10.5.1.1.3 State event matrix

State event matrix of the Link Producer is shown in Table 288.

Table 288 – State event matrix of link producer

Event	State	
	Non-existent	**Running**
Create	Transition to **Running**	Error
Delete	Error	Transition to **Non-existent**
Send	Error	1) Produce TPDU

10.5.1.2 Link consumer

10.5.1.2.1 Link consumer states

The defined states and their descriptions of the Link Consumer are listed in Table 289 below:

Table 289 – Link consumer states

State	Description
Non-existent	The Link Producer does not exist.
Running	The Link Producer has been created and is running.

10.5.1.2.2 State transition diagram

Figure 59 shows the state transition diagram of a link consumer.

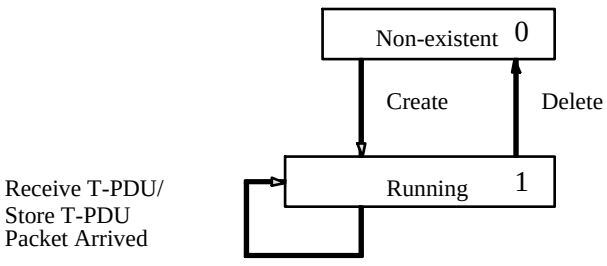


Figure 59 – State transition diagram for a link consumer

10.5.1.2.3 State event matrix

State event matrix of the Link Producer is shown in Table 290.

Table 290 – State event matrix of link consumer

Event	State	
	Non-existent	**Running**
Create	Transition to **Running**	Error
Delete	Error	Transition to **Non-existent**
Receive TPDU	Error	Store TPDU Packet Arrived

10.5.2 Functions used by DMPM

This fieldbus does not any specified functions.

10.6 Data-link Layer service selection

This fieldbus supports only the Data-link layer service whose primitives are noted in 10.4.3.

11 DLL mapping protocol machine 2 (DMPM 2)

11.1 General

Clause 11 defines the requirements for mapping the transmission of TPDU's over an Ethernet-TCP/UDP/IP-based network, instead of the Type 2 data-link layer, using the TCP/UDP/IP protocol suite and the encapsulation protocol defined in 4.3.1.

11.2 Mapping of UCMM PDUs

11.2.1 General

UCMM PDUs shall be transmitted over a TCP/IP connection, using the encapsulation protocol defined in 11.8 and 4.3.1 (SendRRData command).

A UCMM request shall be formatted as shown in Table 291.

Table 291 – UCMM request

Structure	Field name		Data type	Field value
Encapsulation header	Command		UINT	SendRRData (0x6F)
	Length		UINT	Length of command specific data portion
	Session handle		UDINT	Handle returned by RegisterSession
	Status		UDINT	0
	Sender Context		ARRAY of 8 USHORT	Chosen by sender
	Options		UDINT	0
Command specific data	Interface handle		UDINT	0
	Timeout		UINT	Any value (ignored by target)
	Encapsulated packet	Item count	UINT	2 (indicates 1 address item, 1 data item)
	(in the Common Packet Format)	Address Type ID	UINT	0 (Null Address Item)

Structure	Field name		Data type	Field value
		Address length	UINT	0
		Data Type ID	UINT	0x00B2 (Unconnected Data Item)
		Data length	UINT	Length of the next field in octets (length of the MR request packet)
		MR request packet	ARRAY of USINT	This field contains a Type 2 Message Router request packet

A UCMM reply shall be formatted as shown in Table 292.

Table 292 – UCMM reply

Structure	Field name		Data type	Field value
Encapsulation header	Command		UINT	SendRRData (0x6F)
	Length		UINT	Length of command specific data portion
	Session handle		UDINT	Handle returned by RegisterSession
	Status		UDINT	0
	Sender Context		ARRAY of 8 USHORT	Copied from the corresponding UCMM request
	Options		UDINT	0
Command specific data	Interface handle		UDINT	0
	Timeout		UINT	Any value (ignored by receiver)
	Encapsulated packet (in the Common Packet Format)	Item count	UINT	2 (indicates 1 address item, 1 data item)
		Address Type ID	UINT	0 (Null Address Item)
		Address length	UINT	0
		Data Type ID	UINT	0x00B2 (Unconnected Data Item)
		Data length	UINT	Length of the next field in octets (length of the MR response packet)
		MR response packet	ARRAY of USINT	This field contains a Type 2 Message Router response packet

NOTE Subclause 4.3.2 defines the valid format for the SendRRData and other encapsulation commands.

The maximum size of the MR request and MR response packet items in the SendRRData command is 504 octets (the maximum size corresponding to Type 2 data link layer), to ensure that a UCMM message can traverse all the links in a Type 2 network path. Devices may support the Large_Forward_Open service to allow more efficient access to large application data sizes.

When a target receives a SendRRData greater than the maximum size, it shall return an error response with encapsulation status code 0x65 (target received a message of invalid length). Status code 0x03 is allowed for existing implementations but shall be considered to be “deprecated”. New implementations shall use 0x65.

11.2.2 Common requirements for Connection Manager PDU's

11.2.2.1 General

Subclause 11.2.2 contains the specific requirements for Connection Manager parameters in case of TCP/IP encapsulation (Connection Manager parameters are fully described in 4.1.6).

11.2.2.2 Connection type

The connection type shall be NULL, MULTICAST, or POINT2POINT. The MULTICAST connection type shall be supported only for transport class 0 and class 1 connections.

11.2.2.3 Priority

The priority shall be LOW, or HIGH, SCHEDULED or URGENT, with URGENT being the highest priority. Subclause 11.7 defines Quality of Service (QoS) behavior with respect to the different Type 2 priority levels.

11.2.2.4 Trigger type

The trigger type shall be CYCLIC, CHANGE_OF_STATE, or APPLICATION. Transport class 0 and class 1 connections that use CHANGE_OF_STATE triggering shall use the production inhibit time segment (see 4.1.9.5).

11.2.2.5 Connection size

The connection size shall be no larger than 65 511 octets.

NOTE The Forward_Open request limits the connection size to 511 octets; however, the optional Large_Forward_Open allows larger connection sizes.

11.2.2.6 Connection request timeout

To reliably establish a connection that extends onto a TCP/IP link, the connection request time-out shall be large enough to allow the connection to be established, which could involve resolving a host name, or going through multiple gateways.

Because of the large variation in connection request processing over TCP/IP, Type 2 routers in the connection path shall not subtract anything from the connection request timeout.

11.2.2.7 Connection path

The link address portion of a TCP/IP connection path segment shall be encoded within a port segment as specified in 4.1.9.3.

11.2.2.8 Network Connection ID

The Network Connection ID shall be a 32-bit identifier meaningful to the device that chooses it. The Network Connection ID need not be subdivided into any specific fields.

In general, the consuming device selects the Network Connection ID for a point-to-point connection, and the producing device selects the Network Connection ID for a multipoint connection. Table 293 shows which device, Target or Originator, shall choose the T⇒O and O⇒T Network Connection IDs.

Table 293 – Network Connection ID selection

Connection type	Which network Connection ID	Who chooses Connection ID
Point-to-point	Originator => Target	Target
	Target => Originator	Originator
Multicast	Originator => Target	Originator
	Target => Originator	Target

The Network Connection ID shall not be reused until the connection has been closed or has timed out. When a device restarts, it shall not reuse Network Connection IDs from previously opened connections until those connections have been closed or have timed out. A specific connection ID shall not be reused so long as there is the possibility that packets with that connection ID are present in the network.

11.2.3 Forward_open PDU for class 2 and class 3 connections

The Forward_open PDU for class 2 and class 3 connections shall be sent over a TCP connection using the SendRRData command defined in 4.3.1.

Sockaddr Info items included within a Forward_open request or Forward_open_response PDU that establishes a class 2 or class 3 connection shall be ignored. When Sockaddr Info items are included, it is recommended that the sender sets the sin_port and sin_addr fields to 0.

11.2.4 Forward_open for class 0 and class 1 connections

11.2.4.1 Use of TCP

The Forward_Open PDU for transport class 0 and class 1 connections shall be sent over a TCP connection using the SendRRData command defined in 4.3.1.

11.2.4.2 General use of Sockaddr Info items

As part of the Forward_Open dialog, the producer and consumer shall exchange the UDP port numbers and IP multicast address (for multipoint connections) necessary to send the transport class 0 and class 1 connected data. The Sockaddr Info item defined in 4.3.1 shall be used to encode the UDP port numbers and IP multicast address. The inclusion and use of a Sockaddr Info item varies depending on whether the connection is multipoint or point-to-point, and whether the connection originator or the connection target is the multipoint producer.

11.2.4.3 Sockaddr Info item placement and errors

The Sockaddr Info item(s) shall be placed after the Forward_Open request and/or Forward_Open reply data in the SendRRData command/reply. It shall be considered an error if a Sockaddr Info item is not present for a multipoint connection, does not have the correct sin_family value of AF_INET = 2, or specifies field values which are illegal for the intended usage.

The receiver of a Forward_open_request shall return a Forward_open_response with a status code 0x01 and extended status 0x205 (parameter error in unconnected service) for Sockaddr Info items containing errors specified in the previous paragraph. The receiver of a Forward_open response containing any Sockaddr Info items with errors shall consider the entire response to be in error.

11.2.4.4 Use of Sockaddr Info item for multipoint connections

For multipoint connections, the multipoint producer shall choose an IP multicast address to which to send the connected data. The port number shall be the registered UDP port number (0x08AE) assigned by the IANA. The multipoint consumer shall receive the connected data on

the registered port number. A Sockaddr Info item shall be sent with the Forward_Open request (if the O⇒T Connection Type is multipoint), or with the Forward_Open response (if the T⇒O Connection Type is multipoint). The chosen IP multicast address shall be encoded via the sin_addr field of the Sockaddr Info item. The sin_port field of the Sockaddr Info item shall be set to 0x08AE and treated by the receiver as “don’t care”.

11.2.4.5 Use of Sockaddr Info item for point-to-point connections

For point-to-point connections, the point-to-point consumer shall choose a UDP port number on which it will receive the connected data. The point-to-point producer shall send the connected data to the port number chosen by the point-to-point consumer. It is recommended that the consumer use the registered port number (0x08AE), however the consumer may alternatively choose a different port number.

NOTE Using a port number other than 0x08AE has potential implication for Quality of Service (QoS) configuration and management in infrastructure devices. Switches that have been configured to prioritize packets with port number 0x8AE may not prioritize Type 2 packets with a different port number.

If the point-to-point consumer chooses a port number that is different than 0x08AE, then the point-to-point consumer shall send a Sockaddr Info item indicating the chosen port number. The Sockaddr Info item shall be sent with the Forward_open request (if the T⇒O Connection Type is point-to-point), or with the Forward_open response (if the O⇒T Connection Type is point-to-point). The sin_port field of the Sockaddr Info item shall contain the port number selected by the consumer.

If the point-to-point consumer elects to use the registered port 0x08AE to receive connected data, the consumer is not required to send a Sockaddr Info item. The consumer may optionally include a Sockaddr Info item as described in the previous paragraph, with sin_port field containing the registered port number.

For a point-to-point connection the sin_addr field of the Sockaddr Info item shall be treated as a “don’t care” by the receiver of the Forward_open request or Forward_open response. It is recommended that the sender sest the sin_addr field to zero.

11.2.4.6 Usage summary of Sockaddr Info for class 0 or class 1 connections

Table 294 shows the usage of Sockaddr Info items in transport class 0 and 1 Forward_Open request and Forward_Open response. In the cases where the item is “ignored if present”, devices shall not check the contents of the Sockaddr Info item for validity.

Table 294 – Sockaddr Info usage

Connection type	Service	Sockaddr Info items
Point-Point O⇒T Multipoint T⇒O	Forward_Open request	O⇒T ignored if present T⇒O ignored if present
	Forward_Open response	O⇒T item optional (registered port number used if item is not present) T⇒O item required
Multipoint O⇒T Point-Point T⇒O	Forward_Open request	O⇒T item required T⇒O item optional (registered port number used if item is not present)
	Forward_Open response	O⇒T ignored if present T⇒O ignored if present
Point-Point O⇒T Point-Point T⇒O	Forward_Open request	O⇒T ignored if present T⇒O item optional (registered port number used if item is not present)
	Forward_Open response	O⇒T item optional (registered port number used if item is not present) T⇒O ignored if present
Multipoint O⇒T Multipoint T⇒O	Forward_Open request	O⇒T item required T⇒O ignored if present
	Forward_Open response	O⇒T ignored if present T⇒O item required

11.2.4.7 Mapping connections to IP multicast addresses

It is recommended, though not required, that producers use a unique IP multicast address for each active multipoint connection.

NOTE Depending upon the implementation, this can reduce the amount of connection screening on the part of the consumer. It also allows the consumer to more evenly service incoming connected data from multiple connections.

Since a unique IP multicast addresses per multipoint connection is not required, consumers shall be able to handle the situation in which packets from multiple multipoint connections are being sent to the same IP multicast address. Consumers shall be able to screen the incoming packets based on the Connection ID and source IP address (see 11.3.4 for screening requirements).

11.2.4.8 Completing the multipoint connection (informative)

After receiving the Forward_Open response the consuming Ethernet devices should join the desired IP Multicast Group in order to receive the IP Multicast datagrams. The exact method for doing this depends on the TCP/IP application programming interface in use on the device.

11.2.4.9 IP multicast scoping and address allocation

11.2.4.9.1 General

Subclauses 11.2.4.9.2 to 11.2.4.9.5 addresses two issues that shall be considered when implementing multipoint connections on Ethernet:

- IP multicast scoping refers to the practice of limiting how widely a given multicast datagram is propagated across the network;
- IP multicast address allocation refers to the problem of how applications select IP multicast addresses that are used to send and receive IP multicast datagrams.

NOTE Once the IP multicast allocation architecture becomes standard and is deployed, requirements detailed in 11.2.4.9.2 to 11.2.4.9.5 can be updated as appropriate.

11.2.4.9.2 IP multicast scoping practices (informative)

In general, most currently deployed networks use the practice of “TTL scoping” in conjunction with router and/or switch configuration to confine multicast traffic to desired network boundaries.

TTL scoping refers to the practice of using the “Time to Live” (TTL) field in the IP header to limit the number of network hops over which the multicast packet is propagated. When sending an IP multicast datagram, a host can set the TTL field in the IP header to an appropriate value based on how widely the datagram should be propagated. As the datagram is routed through the network, each hop decrements the TTL field. Routers can be configured with TTL thresholds such that they will not forward a packet unless the TTL is greater than the threshold.

A multicast datagram with an initial TTL of 1 limits the datagram to the local subnet. Other common TTL values are 16 for multicast within a site and 64 for multicast within a region.

In addition to TTL scoping, multicast routing protocols and other methods are commonly used to control the propagation of multicast traffic. Routers commonly support multicast protocols such as PIM, DVMRP. Switches that implement “IGMP snooping” can limit the multicast packets sent on a port to only those multicast addresses for which the end device has issued an IGMP membership message. Configuration of switches and routers is usually done by knowledgeable staff.

11.2.4.9.3 IP multicast address allocation practices (informative)

The entire IP multicast address space is 224.0.0.0 to 239.255.255.255. The Internet Assigned Numbers Authority (www.iana.org) is responsible for allocation of the IP multicast address space. IP multicast addresses have been assigned to particular organizations, and for particular protocols. In addition there is a large block of IP multicast addresses allocated for "administratively scoped" multicast, from which applications may allocate addresses, and for which a suite of allocation and scoping protocols are being developed by the Internet Engineering Task Force (IETF). The administratively scoped range is from 239.0.0.0 through 239.255.255.255 (and is further partitioned into additional ranges).

Unfortunately at present there are no widely deployed standard mechanisms for allocating and assigning multicast addresses to applications. For example, when a network administrator deploys a video streaming application, the application will have its own specific mechanism for assigning IP multicast addresses.

11.2.4.9.4 Multicast scoping for Type 2

By default, Type 2 devices shall use a TTL equal to 1 for transport class 0 and 1 multicast packets. The use of a TTL value of 1 prevents multicast packets from propagating beyond the local subnet. When TTL is equal to 1, both the Type 2 producer and consumer shall be on the same subnet.

Type 2 devices are strongly encouraged to support the explicit configuration of the TTL value for IP multicast packets. If supported, devices shall use the TCP/IP Interface object (class 0xF5) as the mechanism to configure the TTL value. When a TTL value greater than 1 is configured, then the producer and consumer may be on different subnets.

If the TTL value has not been configured to be greater than 1 and if a multipoint connection request is received from an originator on a different subnet, then the device shall return general status 0x01 and extended status 0x813 in the Forward_Open response.

When the TTL value is explicitly configured, it shall be used for all Type 2 multicast packets.

11.2.4.9.5 Multicast address allocation for Type 2

11.2.4.9.5.1 General

Two mechanisms are defined for allocation of IP multicast addresses used for Type 2 multicast packets: using an algorithm based on the device's IP address, and explicit configuration via the TCP/IP Interface object. Type 2 devices shall implement the algorithm-based method by default. Devices are also strongly encouraged to implement the method for explicit configuration of multicast addresses. Both methods are described below.

11.2.4.9.5.2 Allocation algorithm based on the device's IP address

The overall IP multicast address range shall be the Organizational Local Scope, and shall start at 239.192.1.0. Each device shall use a block of (at most) 32 multicast addresses from this range. Each device shall calculate the block of multicast addresses via an algorithm, described further below, that uses the Host Id portion of the device's IP address.

A device's Host Id shall be determined by applying the subnet mask to the device's IP address. If a subnet mask is configured for the device, the subnet mask shall be applied to the IP address to determine the Host Id. If no subnet mask is in use, then the class of the device's IP address shall be used to determine the Host Id (e.g., for Class C addresses 8 bits of Host Id shall be used, for Class B addresses 16 bits of host id shall be used, and for Class A addresses 24 bits of Host Id shall be used).

In order to keep the IP multicast addresses within the IPv4 Organization Local Scope, and to put a reasonable bounds on the number of multicast addresses in use, devices shall use at

most the low-order 10 bits of the Host Id in generating the range of multicast addresses. This allows for 1 024 unique Host Ids.

The following pseudo-code shows the algorithm to determine the device's starting and ending multicast addresses:

```
Type2_Mcast_Base_Addr = 0xEFC00100 // 239.192.1.0 is the starting address
Type2_Host_Mask = 0x3FF          // 10 bits of host id

if Subnet_mask configured then
   Netmask = Subnet_mask
else
   if IP_address is Class A then
      Netmask = 255.0.0.0
   else if IP_address is Class B then
      Netmask = 255.255.0.0
   else if IP_address is Class C then
      Netmask = 255.255.255.0
end_else

Host_id = IP_addr & (~Netmask)
Mcast_index = Host_id - 1
Mcast_index = Mcast_index & (Type2_Host_Mask)

Mcast_start_addr = Type2_Mcast_Base_Addr + (Mcast_index * 32)
Mcast_end_addr = Mcast_start_addr + 31
```

Table 295 shows example multicast assignments.

Table 295 – Example multicast assignments

Subnet mask	IP address	Multicast addresses
None configured (8 bits of Host Id used, since 192.168.x.x is Class C)	192.168.1.1	239.192.1.0 to 239.192.1.31
	192.168.1.2	239.192.1.32 to 239.192.1.63
	192.168.1.3	239.192.1.64 to 239.192.1.95
255.255.248.0 (11 bits of Host Id)	10.10.16.1	239.192.1.0 to 239.192.1.31
	10.10.16.2	239.192.1.32 to 239.192.1.63
	10.10.16.3	239.192.1.64 to 239.192.1.95
	10.10.20.0 (Host Id = 1 024; lower 10 bits all 0)	239.192.128.224 to 239.192.128.255 (this is the highest multicast address range that would result using the algorithm)

Since there are a finite number of unique Host Ids, it is possible for different devices to produce data using the same multicast address. Consequently, devices that receive packets on Type 2 multicast connections shall screen the incoming packets based on the Connection Id as well as the IP address of the sending device. This is described in 11.3.4.

When the device's IP address or subnet mask changes, the IP multicast addresses generated by the algorithm also shall change accordingly.

11.2.4.9.5.3 Explicit configuration of IP multicast addresses

IP multicast addresses are configured via the TCP/IP Interface object (class 0xF5). The user (or software) can configure a starting multicast address and number of multicast addresses to allocate. The configured multicast addresses shall then be used for Type 2 multicast packets.

Type 2 devices shall at least implement the algorithmic method for allocating IP multicast addresses, and are encouraged to implement both methods. If both methods are implemented, the algorithmic method shall be the default “out of box” method.

11.3 Mapping of transport class 0 and class 1 PDUs

11.3.1 UDP datagrams

PDUs for transport class 0 and class 1 connections shall be transmitted using UDP. PDUs for multipoint connections shall be transmitted using IP multicast.

The packet shall be formatted as shown in Table 296. Note that the packets use the Common Packet Format defined in 4.3.3, but without the encapsulation header.

Table 296 – UDP data format for class 0 and class 1

Field name	Type	Value
Item Count	UINT	2
Type ID	UINT	0x8002 (Sequenced Address Type)
Length	UINT	8
Address Data	UDINT	Connection ID (from Forward_open response)
	UDINT	Encapsulation Sequence Number. This is different from the sequence count in the transport class 1 packet (see 11.3.3)
Type ID	UINT	0x00B1 (Connected Data Type)
Length	UINT	Number of octets in PDU to follow
Data	ARRAY of octets	Transport class 0 or class 1 PDU

11.3.2 No linkage with TCP connections

In order to open a transport class 0 or 1 connection, a TCP connection and an encapsulation session shall first be established. The TCP connection is used to send the Forward_Open request and receive the Forward_Open response. Once the TCP connection is opened, and transport class 0 and 1 connections are established, it is recommended that Type 2 devices leave the TCP connection open. If the TCP connection is left open, it is then available for subsequent communications such as a Forward_Close or other explicit messages.

Although it is recommended that devices leave the TCP connection open, there shall be no linkage between the TCP connection used to open a transport class 0 or class 1 connection and the resulting class 0 or class 1 connection. If a TCP connection closes, the closing of the TCP connection shall not cause the target or the originator to close any corresponding transport class 0 or class 1 connections.

11.3.3 Class 0 and class 1 packet ordering

NOTE 1 By definition class 0 and class 1 transports do not detect out-of-order packets. For class 0, every packet is considered to be new data. For class 1, only duplicate data is detected. A received packet is considered to be new data whenever the sequence count is different (either greater or lesser) than the previous packet's sequence count).

NOTE 2 When using UDP to transport Class 0 and class 1 connected data, there is no guarantee that packets arrive in the same order that they were sent. When both sender and receiver are on the same subnet, packets

typically arrive in order. However, when going through routers, when there are multiple paths that a packet could take, it is possible for packets to arrive out of order.

For class 0 and class 1 connections over Ethernet, devices shall maintain an encapsulation sequence number in the data portion of the UDP datagram defined in 11.3.1. The encapsulation sequence number shall be maintained per connection. Each time an Ethernet device sends a Class 1 packet, it shall increment the encapsulation sequence number for that connection. If the receiving Ethernet device receives a packet whose encapsulation sequence number is less than the previously received packet, the packet with the smaller encapsulation sequence number shall be discarded.

The sequence number shall be operated on with modular arithmetic to deal with sequence rollover.

NOTE 3 Dealing with 32-bit sequence numbers is described in IETF RFC 793 (the TCP definition).

11.3.4 Screening incoming connected data

Ethernet devices that receive class 0 and class 1 connected data shall screen incoming packets based on the Connection ID and IP address of the sending device. This is necessary for the following reasons.

- For multipoint connections, there is no guaranteed mechanism to prevent multiple devices from using the same IP multicast address. Consequently, a device could receive (bogus) multipoint connected data from a device with which it has not established a connection.
- For multipoint connections, a device is allowed to use the same IP multicast address for multiple class 0 and class 1 multipoint connections.
- To prevent Connection ID conflicts.

When a class 0 or class 1 connection is established, the target and originating Ethernet devices shall record the Connection ID on which they will receive connected data, coupled with the IP address of the device at the other end of the connection. When a device receives connected data, it shall confirm that the Connection ID is valid for the IP address of the sending device. If not, the packet shall be discarded.

11.4 Mapping of transport class 2 and class 3 PDU's

PDUs for transport class 2 and class 3 connections shall be transmitted over a TCP connection using the encapsulation protocol defined in 11.8 and 4.3.1 (SendUnitData command).

The connected data shall be formatted as shown in Table 297.

Table 297 – Transport class 2 and class 3 connected data

Structure	Field name	Data type	Field value
Encapsulation header	Command	UINT	SendUnitData (0x70)
	Length	UINT	Length of command specific data portion
	Session handle	UDINT	Handle returned by RegisterSession
	Status	UDINT	0
	Sender Context	ARRAY of 8 USHORT	Any value (ignored by target)
	Options	UDINT	0
Command specific data	Interface handle	UDINT	0

Structure	Field name		Data type	Field value
	Timeout		UINT	Any value (ignored by target)
	Encapsulated packet (in the Common Packet Format)	Item count	UINT	2 (indicates 1 address item, 1 data item)
		Address Type ID	UINT	0x00A1 (Connected Address Item)
		Address length	UINT	4
		Address data	UDINT	Connection ID from Forward_Open/ Forward_Open_Reply
		Data Type ID	UINT	0x00B1 (Connected Data Item)
		Data length	UINT	Length of the next field in octets (i.e., length of the transport class 2/3 PDU, including the sequence count)
		Data	ARRAY of USINT	The transport class 2/3 PDU (including the sequence count)

Multiple Type 2 connections may be sent over a single TCP connection. An implementation need not support a specific number of connections per TCP connection. An implementation may impose an upper bound if it chooses.

Because of the full-duplex nature of TCP, the O2T and T2O link connections shall use the same TCP connection. However if a target subsequently originates a Type 2 connection, then it shall be considered an originator, and a different TCP connection shall be used.

NOTE This standard defines no requirements for management of the TCP connection, such as inactivity timeouts, or closing the TCP connection when all native connections are closed. However, implementations are free to implement these.

Targets and originators shall close any Type 2 transport class 2 or 3 connections when the corresponding originating TCP connection is closed.

11.5 Mapping of transport classes 4 to 6

The encapsulation protocol defined in 11.8 and 4.3.1 shall not be used to encapsulate PDUs for transport classes 4, 5 and 6.

11.6 IGMP Usage

11.6.1 Background (informative)

The Internet Group Management Protocol (IGMP) is a standard protocol used by hosts to report their IP multicast group memberships and must be implemented by any host that wishes to receive IP multicast datagrams. IGMP messages are used by multicast routers to learn which multicast groups have members on their attached networks. IGMP messages are also used by switches capable of supporting “IGMP snooping” whereby the switch listens to IGMP messages and only sends the multicast packets to ports that have joined the multicast group.

There are three versions of IGMP:

- IGMP V1 is defined in IETF RFC 1112.
- IGMP V2 is defined in IETF RFC 2236.
- IGMP V3 is defined in IETF RFC 3376.

IETF RFC 2236 and IETF RFC 3376 discuss host and router requirements for interoperation with older IGMP versions.

Since Type 2 devices make extensive use of IP multicast for transport class 0 and 1 connections, consistent IGMP usage by Type 2 devices is essential in order to create well-functioning Type 2 application networks.

11.6.2 IGMP Membership Report messages

Type 2 devices shall issue a Membership Report message (V1, V2 or V3 as appropriate) when opening a Type 2 connection on which they will receive multicast packets. Specifically, devices shall adhere to the following behavior.

- When the T=>O Connection Type is multipoint (originator is multipoint consumer), the originator shall issue a Membership Report upon receipt of a successful Forward_Open response. The Membership Report shall include the IP multicast address as communicated in the Forward_Open response.
- When the O=>T Connection Type is multipoint (target is multipoint consumer), the target shall issue a Membership Report upon sending a successful Forward_Open response. The Membership Report shall include the IP multicast address as communicated in the Forward_Open.

If the device has already issued a Membership Report for the IP multicast address (e.g. if the multicast address is being used with an existing connection), the device may, but is not required to, issue another Membership Report.

Devices shall also send Membership Report messages in response to Membership Query messages, per the IGMP RFCs.

11.6.3 IGMP Leave Group messages

Devices that support IGMP V2 shall issue a Leave Group when all the Type 2 connections associated with a consuming IP multicast address have either closed or timed out. Specifically, devices shall adhere to the following behavior:

- When the T=>O Connection Type is multipoint (originator is multipoint consumer), the originator shall issue a Leave Group upon receipt of a successful Forward_Close response if the originator has no other open connections consuming on that IP multicast address.
- When the O=>T Connection Type is multipoint (target is multipoint consumer), the target shall issue a Leave Group upon sending a successful Forward_Close response if the target has no other open connections consuming on that IP multicast address.
- In the event of a connection timeout, the multipoint consumer (whether target or originator) shall issue a Leave Group message if the multipoint consumer has no other connections consuming on that IP multicast address.

11.7 Quality of Service (QoS) for CP 2/2 messages

11.7.1 Overview

Quality of Service (QoS) is a general term for mechanisms that treat traffic streams with different relative priorities or other delivery characteristics. Standard QoS mechanisms include IEEE 802.1D/IEEE 802.1Q (Ethernet frame priority) and Differentiated Services (DiffServ) in the TCP/IP protocol suite.

Within the CPF 2 and CP 2/2 application context, QoS is especially important for time sensitive applications where packet delivery will affect application stability.

CP 2/2 defines requirements and recommendations for how CP 2/2 devices use two standard QoS mechanisms: IEEE 802.1D/IEEE 802.1Q and Differentiated Services. The following summarizes the QoS mechanisms defined for CP 2/2 devices.

- The overall approach is for CP 2/2 end devices to mark CP 2/2 packets with priority values, using the standard mechanisms mentioned above. Marking packets with priority enables infrastructure devices (e.g., switches and routers) to better differentiate CP 2/2 traffic.
- For CPF 2 transport class 0 and 1 connections (i.e., UDP-based), there is a defined mapping of CPF 2 priorities to IEEE 802.1D priorities and DiffServ Code Points (see 11.7.4).
- For UCMM and CPF 2 transport class 2 and 3 connections (i.e., TCP-based), and all other CP 2/2 encapsulation messages (both TCP-based and UDP-based) there is a defined DiffServ Code Point, and a defined IEEE 802.1D priority value.
- For PTP (IEC 61588) messages, there are DiffServ Code Points and IEEE 802.1D priority values corresponding to the two different types of PTP messages.
- It is strongly recommended that devices by default mark CPF 2 and PTP messages with DSCP values.
- In addition to DSCP, devices may optionally support sending and receiving IEEE 802.1Q tagged frames. If supported, sending tagged frames shall be disabled by default in order to prevent device interoperability problems. When IEEE 802.1Q tagging is desired, the end user shall explicitly enable the sending of IEEE 802.1Q frames, and shall ensure that both the sending and receiving devices support tagged frames.
- The QoS Object provides a means to configure DSCP values, and a means to enable/disable sending of IEEE 802.1Q tagged frames (see IEC 61158-4-2, 7.10).
- There are no requirements for devices to mark traffic other than CPF 2 or IEC 61588. Devices may do so, depending on their implementation capabilities.
- At present there are no specific implementation requirements for end devices to internally differentiate traffic with different priorities. Traffic differentiation in end devices is an area of future development. CP 2/2 implementations are at a minimum recommended to give higher priority to processing CP 2/2 implicit messages over explicit messages. Further differentiation based on the above QoS mechanisms, where possible, is also recommended.

The following references are applicable to QoS for CP 2/2 devices: IEEE 802.1D, IEEE 802.1Q, IETF RFC 2474, IETF RFC 2475, IETF RFC 2597, IETF RFC 2873, IETF RFC 3140, IETF RFC 3246, IETF RFC 4594.

11.7.2 DSCP format

Differentiated Services (DiffServ) is a model for specifying the relative priority of traffic based on the type of service (ToS) field of an IPv4 packet. The model is defined in IETF RFC 2475. DiffServ allows nodes to route packets based on class of traffic as defined by the DiffServ Codepoint (DSCP) and the defined Per-Hop Behavior (PHB) characteristics.

Figure 60 shows the DS field in the IP header.

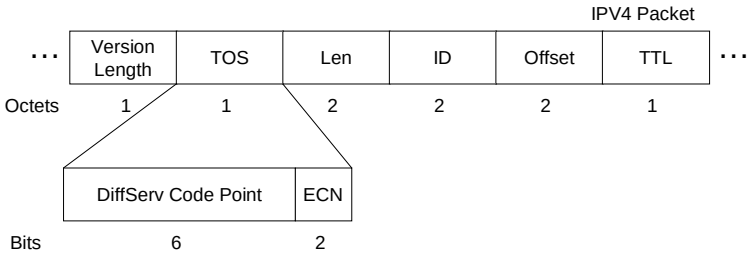


Figure 60 – DS field in the IP header

When setting the DSCP value in the IP header, if placed directly in the ToS field, it shall be shifted left 2 bits.

11.7.3 IEEE 802.1D/Q format

IEEE 802.1Q defines an Ethernet frame format that allows inclusion of VLAN ID and priority. The IEEE 802.1Q frame has EtherType of 0x8100 and a 4-octet prefix between the Source and Type fields of the frame. The tagged frame defines a 3-bit field to specify 8 priority levels, further specified in IEEE 802.1D. Priority 7 is the highest. Priority 0 is the lowest.

Figure 61 shows the format of an IEEE 802.1Q frame.

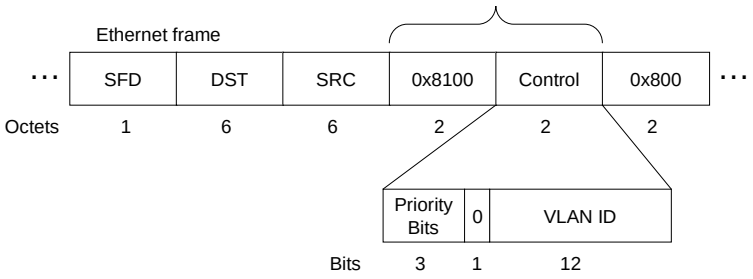


Figure 61 – IEEE 802.1Q tagged frame

11.7.4 Mapping CPF 2 traffic to DSCP and IEEE 802.1D

Table 298 defines the default DSCP and IEEE 802.1D priority mappings for CP 2/2 and CP 2/2.1 (IEC 61588) traffic.

Table 298 – Default DSCP and IEEE 802.1D mapping

Traffic type	CPF 2 priority	DSCP	IEEE 802.1D Priority[a]	CPF 2 traffic usage (recommended)
PTP Event (IEC 61588)	n/a	59 (‘111011’)	7	n/a
PTP General (IEC 61588)	n/a	47 (‘101111’)	5	n/a
CPF 2 class 0/1	Urgent (3)	55 (‘110111’)	6	CIP Motion
	Scheduled (2)	47 (‘101111’)	5	Safety I/O, I/O
	High (1)	43 (‘101011’)	5	I/O
	Low (0)	31 (‘011111’)	3	No recommendation at present
CPF 2 UCMM CPF 2 class 2/3 All other CP 2/2 encapsulation messages	All	27 (‘011011’)	3	CPF 2 messaging

[a] Sending IEEE 802.1Q tagged frames is disabled by default.

NOTE 1 In Table 298, the IEEE 802.1D and DSCP values for transport class 0 and class 1 messages are based on the CPF 2 priority (indicated in the Forward Open service). PTP and CPF 2 explicit messages use IEEE 802.1D and DSCP values independent of CPF 2 priority.

The default DSCP values and IEEE 802.1Q tagged frame enable/disable may be changed via the QoS Object (see IEC 61158-4-2, 7.10).

NOTE 2 The default DSCP values are “local use” values, as defined in IETF RFC 2474.

11.7.5 CP 2/2 usage of DSCP

When marking CP 2/2 traffic with DSCP values, CP 2/2 devices shall use the values as configured in the QoS Object (default values are shown in Table 298). Usage of the ECN field in the IP header by CP 2/2 end devices is not currently defined and shall be set to 0.

Notice that when DSCP marking is supported, all outgoing packets for established TCP encapsulation connections shall be marked with the appropriate DSCP value, including ACK-only packets. It is recommended that all outgoing packets involved in the opening and closing of these connections also be marked (SYN, FIN, RST, and associated ACKs).

Devices are strongly encouraged to support marking CP 2/2 packets with DSCP values. When supported, the default behavior shall be to mark packets.

All CP 2/2 devices shall support receiving packets with non-zero DSCP values since this is an Internet Protocol requirement (see IETF RFC 791 and IETF RFC 1122). Ethernet infrastructure devices (e.g., switches and routers) can potentially alter DSCP values. Receiving devices shall not assume or otherwise check that incoming DSCP values are the same as in Table 298, or are the same values as sent from the sending device.

NOTE This behavior is consistent with modifications to TCP (IETF RFC 793) as described by IETF RFC 2873.

11.7.6 CP 2/2 usage of IEEE 802.1D/Q

When sending traffic with IEEE 802.1Q tagged frames, CP 2/2 devices shall use the priority values specified in Table 298. The VLAN ID shall be set to 0, unless a specific VLAN ID for the device has been configured by means not covered by this specification. When receiving IEEE 802.1Q tagged frames, devices shall not require that the VLAN ID be set to 0.

When sending IEEE 802.1Q tagged frames, devices shall also set the corresponding DSCP value in the IP header.

When IEEE 802.1Q frames are supported, the device shall also support the QoS Object (see IEC 61158-4-2, 7.10). The default behavior shall be to disable sending of IEEE 802.1Q tagged frames, since sending tagged frames by default can result in device interoperability problems.

Notice that when sending traffic with IEEE 802.1Q tagged frames, all outgoing packets for established TCP encapsulation connections shall use the specified IEEE 802.1D priority value, including ACK-only packets. It is recommended that all outgoing packets involved in the opening and closing of these connections (SYN, FIN, RST, and associated ACKs) also use the same IEEE 802.1D priority value.

11.7.7 User considerations with IEEE 802.1D/Q

Some CP 2/2 devices do not support receiving IEEE 802.1Q tagged frames. When IEEE 802.1Q frames are sent to such devices, the frames will be dropped. In addition, some managed switches will drop IEEE 802.1Q tagged frames unless the receiving port is configured to accept them.

In order to prevent interoperability problems, sending IEEE 802.1Q frames is disabled by default for CP 2/2 devices. The end user may elect to enable sending tagged frames when supported, via the QoS Object. The user is ultimately responsible for ensuring that both the sending and receiving devices support IEEE 802.1Q frames, and that network infrastructure is properly configured.

11.8 Management of an encapsulation session

11.8.1 Phases of an encapsulation session

An encapsulation session shall have three phases:

- establishing a session;
- maintaining a session;
- closing a session.

11.8.2 Establishing a session

Session establishment shall proceed according to the following steps:

- the originator shall open a TCP/IP connection to the target, using the reserved port number (0xAF12), or if specified, the port number from the connection path;
- the originator shall send a RegisterSession command to the target (see 4.3.2.2);
- the target shall check the protocol version in the command message to verify it supports the same protocol version as the originator. If not, the target shall return a RegisterSession reply with the appropriate Status field along with the highest supported protocol version;
- the target shall check the options flags in the command to verify that it supports the requested options. If not, the target shall return a RegisterSession reply with the appropriate Status field along with the options it supports;
- the target shall assign a new (unique) Session ID and shall send a RegisterSession reply to the originator

Originators shall register no more than one active session on a single TCP connection.

NOTE A TCP port number, 0xAF12, has been reserved with the Internet Assigned Numbers Authority (IANA) for use by the encapsulation protocol.

11.8.3 Terminating a session

Either the originator or the target may terminate the session. Sessions shall be terminated in either of two ways:

- the originator or target shall close the underlying TCP connection. The corresponding target or originator shall detect the loss of the TCP connection, and shall close its side of the connection;
- the originator or target shall send an UnRegisterSession command (see 4.3.2.3) and shall wait to detect the closing of the TCP connection. The corresponding target or originator shall then close its side of the TCP connection. The sender of the UnRegisterSession shall detect the loss of the TCP connection, then it shall close its side of the connection.

NOTE The second method is preferred since it results in more timely clean up of the TCP connection.

11.8.4 Maintaining a session

Once a session is established, it shall remain established until one of the following occurs:

- the originator or target closes the TCP connection;
- the originator or target issues the UnRegisterSession command;
- the TCP connection is broken.

12 DLL mapping protocol machine 3 (DMPM 3)

This DMPM is specified in IEC 62026-3:2008, Clause 5.

Bibliography

NOTE All parts of the IEC 61158 series, as well as IEC 61784-1 and IEC 61784-2 are maintained simultaneously. Cross-references to these documents within the text therefore refer to the editions as dated in this list of bibliographic references.

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

IEC 61784-1:2014, *Industrial communication networks – Profiles – Part 1: Fieldbus profiles*

IEC 61784-2:2014, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*

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

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

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

IETF RFC 793, *Transmission Control Protocol*, available at <http://www.ietf.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 1: Common Industrial Protocol (CIPTM) – Edition 3.13, November 2012*, available at <http://www.odva.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 2: EtherNet/IPTM Adaptation of CIP – Edition 1.14, November 2012*, available at <http://www.odva.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 3: DeviceNetTM Adaptation of CIP – Edition 1.12, November 2011*, available at <http://www.odva.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 4: ControlNetTM Adaptation of CIP – Edition 1.7, April 2011*, available at <http://www.odva.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 5: CIP SafetyTM – Edition 2.6, November 2012*, available at <http://www.odva.org>

_____

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

RÉSEAUX DE COMMUNICATIONS INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-2: Spécification du protocole de la couche application – Eléments de type 2

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.

L'attention est attirée sur le fait que l'utilisation du type de protocole associé est restreinte par les détenteurs des droits de propriété intellectuelle. En tout état de cause, l'engagement de renonciation partielle aux droits de propriété intellectuelle pris par les détenteurs de ces droits autorise l'utilisation d'un type de protocole de couche avec les autres protocoles de couche du même type, ou dans des combinaisons avec d'autres types autorisées explicitement par les détenteurs des droits de propriété intellectuelle pour ce type.

NOTE Les combinaisons de types de protocoles sont spécifiées dans la CEI 61784-1 et la CEI 61784-2.

La Norme internationale CEI 61158-6-2 a été établie par le sous-comité 65C: Réseaux industriels, du comité d'études 65 de la CEI: Mesure, commande et automation dans les processus industriels.

Cette troisième édition annule et remplace la deuxième édition parue en 2010. Cette édition constitue une révision technique.

Les principales modifications par rapport à l'édition précédente sont les suivantes:

- mise à jour des définitions utilisées par l'objet Time Sync;
- ajout de "membre" et des services spécifiques à un objet en 4.1.2.1, 4.1.8, 4.1.10, 8.2;
- suppression des classes de transport obsolètes 4 à 6 en 4.1.4.6 et de 9.3.9 à 9.3.12;
- clarification des formats d'en-tête de transport en 4.1.4;
- mise à jour du CM et des PDU de MR de 4.1.5 à 4.1.7;
- mise à jour des PDU de l'objet Identity en 4.1.8.2;
- mise à jour des PDU de l'objet Assembly en 4.1.8.4;
- mise à jour des PDU de l'objet Time Sync en 4.1.8.6;
- mise à jour des PDU de l'objet Parameter en 4.1.8.7;
- mise à jour des PDU de l'objet Connection Manager en 4.1.8.8;
- mise à jour des chemins de message et de connexion en 4.1.9;
- mise à jour des codes de classe d'objet en 4.1.10 et des codes d'erreur en 4.1.11;
- mise à jour des types de données en 4.2.4 et 5.2.3;
- mise à jour de la syntaxe abstraite d'encapsulation en 4.3;
- mise à jour du diagramme protocolaire de mapping DLL 2 à l'Article 11;
- corrections rédactionnelles diverses.

Le texte de la présente norme est issu des documents suivants:

FDIS	Rapport de vote
65C/764/FDIS	65C/774/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de la présente norme.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Une liste de toutes les parties de la série CEI 61158, publiées sous le titre général *Réseaux de communication industriels – Spécifications des bus de terrain*, 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.

INTRODUCTION

La présente partie de la CEI 61158 s'inscrit dans une série créée pour faciliter l'interconnexion des composants de systèmes d'automation. Elle renvoie aux autres normes de l'ensemble défini par le modèle de référence de bus de terrain "à trois couches" décrit dans la CEI 61158-1.

Le protocole d'application fournit le service d'application au moyen des services disponibles au niveau de la couche Liaison de données ou de la couche immédiatement inférieure. Le principal objectif de la présente norme est de définir un ensemble de règles de communication, exprimées en termes de procédures, que doivent suivre les entités d'application (Application Entity, AE) homologues au moment de la communication. Ces règles de communication visent à fournir une base saine pour le développement, dans divers buts:

- guider les développeurs et les concepteurs;
- réaliser les essais et acquérir l'équipement;
- dans le cadre d'un accord, admettre des systèmes dans l'environnement de systèmes ouverts;
- améliorer la compréhension des communications en temps critique au sein de l'OSI.

La présente norme traite, en particulier, de la communication et de l'interfonctionnement des capteurs, effecteurs et autres appareils d'automation. Grâce à cette norme associée à d'autres normes des modèles de référence OSI ou de bus de terrain, des systèmes par ailleurs incompatibles peuvent fonctionner ensemble, quelle que soit leur combinaison.

RÉSEAUX DE COMMUNICATIONS INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-2: Spécification du protocole de la couche application – Eléments de type 2

1 Domaine d'application

1.1 Généralités

La couche application de bus de terrain (Fieldbus Application Layer, FAL) procure aux programmes de l'utilisateur un moyen d'accès à l'environnement de communication des bus de terrain. A cet égard, la FAL peut être considérée comme une "fenêtre entre programmes d'application correspondants".

La présente norme fournit des éléments communs pour les communications à temps critique ou non entre des programmes d'application dans un environnement et avec un matériel d'automation spécifiques aux bus de terrain de Type 2. Le terme "à temps critique" signale l'existence d'une fenêtre temporelle dans laquelle est exigée la réalisation d'une ou de plusieurs actions spécifiées, avec un niveau de certitude défini. La non-réalisation des actions spécifiées dans la fenêtre temporelle induit un risque de défaillance des applications qui demandent ces actions, avec les risques afférents pour l'équipement, les installations et éventuellement la vie humaine.

La présente norme spécifie les interactions entre les applications distantes et définit le comportement, visible par un observateur externe, assuré par la couche application de bus de terrain de Type 2, en termes

a) de syntaxe abstraite formelle définissant les unités de données de protocole de couche application, acheminées entre les entités d'application en communication;
b) de syntaxe de transfert définissant les règles de codage qui s'appliquent aux unités de données de protocole de couche application;
c) du diagramme d'états de contexte application définissant le comportement de service application visible entre des entités d'application engagées dans une communication;
d) de diagrammes d'états de relations d'applications définissant le comportement de communication visible entre les entités d'application en communication.

La présente norme vise à définir le protocole mis en place pour

a) définir la représentation filaire des primitives de service définies dans la CEI 61158-5-2 et
b) définir le comportement visible de l'extérieur associé à leur transfert.

La présente norme spécifie le protocole de la couche application de bus de terrain de Type 2, en conformité avec le modèle de référence de base OSI (ISO/CEI 7498-1) et la structure de la couche application OSI (ISO/CEI 9545).

1.2 Spécifications

La présente norme a pour principal objectif de préciser la syntaxe et les caractéristiques du protocole de couche application qui transmet les services de couche application définis dans la CEI 61158-5-2.

Un objectif secondaire consiste à fournir des voies d'évolution à partir des protocoles de communication industriels antérieurs.

1.3 Conformité

La présente norme ne définit pas de mises en œuvre ni de produits particuliers, pas plus qu'elle ne limite les mises en œuvre des entités de couche application dans les systèmes d'automation industriels. La conformité est obtenue par le biais de la mise en œuvre de cette spécification de protocole de couche application.

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).

NOTE Toutes les parties de la série CEI 61158, ainsi que la CEI 61784-1 et la CEI 61784-2 font l'objet d'une maintenance simultanée. Les références croisées à ces documents dans le texte se rapportent par conséquent aux éditions datées dans la présente liste de références normatives.

CEI 61158-1:2014, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 1: Présentation et lignes directrices des séries CEI 61158 et CEI 61784*

CEI 61158-3-2:2014, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 3-2: Définition des services de la couche liaison de données – Eléments de type 2*

CEI 61158-4-2:2014, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 4-2: Spécification du protocole de la couche liaison de données – Eléments de type 2*

CEI 61158-5-2:2014, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 5-2: Définition des services de la couche application – Eléments de type 2*

IEC 61588:2009, *Precision clock synchronization protocol for networked measurement and control systems* (disponible en anglais seulement)

CEI 61784-3-2, *Réseaux de communication industriels – Profils – Partie 3-2: Bus de terrain de sécurité fonctionnelle – Spécifications supplémentaires pour CPF 2*

CEI 61800-7-202, *Entraînements électriques de puissance à vitesse variable – Partie 7-202: Interface générique et utilisation de profils pour les entraînements électriques de puissance – Spécification du profil de type 2*

CEI 62026-3:2008, *Appareillage à basse tension – Interfaces appareil de commande-appareil (CDI) – Partie 3: DeviceNet*

ISO/CEI 7498-1, *Technologies de l'information – Interconnexion de systèmes ouverts – Modèle de référence de base: Le modèle de base*

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

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

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

ISO/CEI 9545, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Structure de la couche Application*

ISO/IEC 10646, *Information technology – Universal Coded Character Set (UCS)* (disponible en anglais seulement)

ISO/CEI 10731, *Technologies de l'information – Interconnexion de systèmes ouverts – Modèle de référence de base – Conventions pour la définition des services OSI*

ISO 639-2, *Codes pour la représentation des noms de langue – Partie 2: code alpha-3*

ISO 11898:1993[1], *Véhicules routiers – Échange d'information numérique – Gestionnaire de réseau de communication à vitesse élevée (CAN)*

IEEE 802.1D-2004, *IEEE standard for local and metropolitan area networks – Media Access Control (MAC) bridges*, disponible à l'adresse http://www.ieee.org

IEEE 802.1Q-2005[1], *IEEE standard for local and metropolitan area networks – Virtual bridged local area networks*, disponible à l'adresse <http://www.ieee.org>

IEEE 802.3-2008: *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications,* disponible à l'adresse <http://www.ieee.org>

IETF RFC 791, *Internet Protocol*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 1035, *Domain Names – Implementation and Specification*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 1112, *Host Extensions for IP Multicasting*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 1117, *Internet Numbers*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 1122, *Requirements for Internet Hosts – Communication Layers*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 1759, *Printer MIB*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 2236, *Internet Group Management Protocol, Version 2*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 2474, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 2475, *An Architecture for Differentiated Services*, disponible à l'adresse <http://www.ietf.org>

[1] Une édition plus récente de cette norme a été publiée, mais seule l'édition citée s'applique.

IETF RFC 2597, *Assured Forwarding PHB Group*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 2873, *TCP Processing of the IPv4 Precedence Field*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 3140, *Per Hop Behavior Identification Codes*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 3246, *An Expedited Forwarding PHB (Per-Hop Behavior)*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 3376, *Internet Group Management Protocol, Version 3*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 4594, *Configuration Guidelines for DiffServ Service Classes*, disponible à l'adresse <http://www.ietf.org>

3 Termes, définitions, symboles, abréviations et conventions

Pour les besoins du présent document, les termes, définitions, symboles, abréviations et conventions suivants s'appliquent.

3.1 Termes et définitions provenant d'autres normes ISO/CEI

3.1.1 Termes et définitions de l'ISO/CEI 7498-1

a) syntaxe abstraite
b) entité d'application
c) processus d'application
d) unité de données de protocole d'application
e) élément de service d'application
f) invocation d'entité d'application
g) invocation de processus d'application
h) transaction d'application
i) contexte de présentation
j) système ouvert réel
k) syntaxe de transfert

3.1.2 Termes et définitions de l'ISO/CEI 9545

a) application-association (association d'applications)
b) application-context (contexte d'application)
c) application context name (nom de contexte d'application)
d) application-entity-invocation (invocation d'entité d'application)
e) application-entity-type (type d'entité d'application)
f) application-process-invocation (invocation de processus d'application)
g) application-process-type (type de processus d'application)
h) application-service-element (élément de service d'application)
i) application control service element (élément de service de contrôle d'application)

3.1.3 Termes et définitions de l'ISO/CEI 8824-1

a) identificateur d'objet
b) type
c) valeur
d) type simple

e) type structuré
f) type de composant
g) étiquette
h) type Boolean (booléen)
i) vrai
j) faux
k) type integer (entier)
l) type bitstring (chaîne binaire)
m) type octetstring (chaîne d'octets)
n) type null (vide)
o) type sequence (séquence)
p) séquence de type
q) type choice (choix)
r) type tagged (étiqueté)
s) any type (n'importe quel type)
t) module
u) production

3.1.4 Termes et définitions de l'ISO/CEI 8825-1

a) codage (d'une valeur de donnée)
b) valeur de donnée
c) octets d'identificateur (le singulier est utilisé dans la présente norme)
d) octet(s) de longueur (le singulier et le pluriel sont utilisés dans la présente norme)
e) octets de contenu

3.2 Termes et définitions de la CEI 61158-5-2

a) relation d'applications
b) client
c) homologue
d) serveur

3.3 Termes et définitions supplémentaires

3.3.1
allouer
prendre une ressource dans une zone commune et affecter la ressource en question à l'usage exclusif d'une entité spécifique

3.3.2
application
fonction ou structure de données pour laquelle des données sont consommées ou produites

3.3.3
objets d'application
classes d'objets multiples qui gèrent et assurent un échange de messages pendant le mode exécution à travers le réseau et à l'intérieur de l'appareil de réseau

3.3.4
attribut
description d'une caractéristique, visible par un observateur externe, d'un objet

Note 1 à l'article: Les attributs d'un objet contiennent des informations sur les portions variables d'un objet. En règle générale, ils fournissent des informations de statut ou régissent l'action d'un objet. Les attributs peuvent également influer sur le comportement d'un objet. Les attributs se répartissent en deux catégories: les attributs de classe et les attributs d'instance.

3.3.5
comportement
indication de la manière dont un objet réagit à des événements particuliers

3.3.6
algorithme d'horloge "meilleure mère"
BMCA, Best Master Clock Algorithm
algorithme exécuté par chaque nœud afin d'identifier l'horloge qui deviendra l'horloge mère d'un sous-réseau et l'horloge grand-mère du domaine

Note 1 à l'article: L'algorithme compare principalement l'attribut priority1, la qualité d'horloge, l'attribut priority2 et l'identité de la source pour identifier la meilleure mère parmi les différentes candidates.

3.3.7
horloge frontière
horloge possédant plusieurs ports PTP (Precision Time Protocol) dans un même domaine, qui maintient l'échelle de temps utilisée dans ce domaine

Note 1 à l'article: Une horloge frontière peut jouer le rôle de source de temps, c'est-à-dire être une horloge mère, ou se synchroniser à une autre horloge, c'est-à-dire être une horloge esclave.

[SOURCE: CEI 61588:2009, 3.1.3, modifiée – deuxième phrase transformée en Note]

3.3.8
appelé
utilisateur de service ou fournisseur de service qui reçoit une primitive "indication" ou une APDU de demande

3.3.9
appelant
utilisateur de service ou fournisseur de service qui lance une primitive "request" ou une APDU de demande

3.3.10
classe
ensemble d'objets représentant le même type de composant du système

Note 1 à l'article: Une classe est une généralisation d'un objet, un modèle qui permet de définir des variables et des méthodes. Tous les objets d'une classe possèdent une forme et un comportement identiques, mais leurs attributs contiennent généralement des données différentes.

3.3.11
attribut de classe
attribut commun à tous les objets d'une classe

3.3.12
code de classe
identificateur unique attribué à chaque classe d'objets

3.3.13
service spécifique à une classe
service défini par une classe d'objets particulière afin de remplir une fonction nécessaire qui n'est assurée par aucun service commun

Note 1 à l'article: Tout objet spécifique à une classe est réservé à l'usage exclusif de la classe d'objets qui le définit.

3.3.14
client
a) objet qui utilise les services d'un autre objet (serveur) pour accomplir une tâche

b) initiateur d'un message auquel un serveur réagit

3.3.15
horloge
nœud participant au protocole PTP, capable de donner une mesure de l'écoulement du temps depuis une époque définie

Note 1 à l'article: Il y a trois types d'horloge dans la CEI 61588:2009: les horloges frontière, les horloges transparentes et les horloges ordinaires.

[SOURCE: CEI 61588:2009, 3.1.4 modifiée – Note différente]

3.3.16
objets de communication
composants qui gèrent et assurent un échange de messages pendant le mode exécution à travers le réseau

EXEMPLES objet Connection Manager (gestionnaire de connexion), objet Unconnected Message Manager (UCMM, gestionnaire de messages non connectés) et objet Message Router (routeur de message).

3.3.17
connexion
liaison logique entre deux objets d'application qui peuvent se trouver au sein du même appareil ou dans des appareils différents

Note 1 à l'article: Les connexions peuvent être soit point à point, soit multipoint.

3.3.18
ID de connexion
CID connection ID
identificateur affecté à une émission qui est associé à une connexion particulière entre producteurs et consommateurs, donnant un nom à un élément spécifique d'information d'application

3.3.19
chemin de connexion
train d'octets qui définit l'objet d'application auquel une instance de connexion s'applique

3.3.20
point de connexion
tampon qui est représenté comme étant une sous-instance d'un objet Assembly (ensemble)

3.3.21
consommer
acte consistant à recevoir des données provenant d'un producteur

3.3.22
consommateur
nœud ou puits qui reçoit des données d'un producteur

3.3.23
application consommatrice
application qui consomme des données

3.3.24
cyclique
qui se reproduit de manière régulière et répétitive

3.3.25
appareil
matériel physique connecté à la liaison

Note 1 à l'article: Un appareil peut contenir plusieurs nœuds.

3.3.26
profil d'appareil
ensemble d'informations et de fonctionnalités dépendantes de l'appareil qui garantit la cohérence entre les appareils similaires appartenant au même type d'appareil

3.3.27
domaine
groupement logique d'horloges synchronisées entre elles à l'aide du protocole, mais pas nécessairement synchronisées aux horloges d'un autre domaine

[SOURCE: CEI 61588:2009, 3.1.7]

3.3.28
nœud d'extrémité
nœud producteur ou consommateur

3.3.29
point d'extrémité
une des entités en communication impliquées dans une connexion

3.3.30
époque
origine d'une échelle de temps

[SOURCE: CEI 61588:2009, 3.1.9]

3.3.31
erreur
divergence entre une valeur ou condition calculée, observée ou mesurée et la valeur ou condition spécifiée ou théoriquement correcte

3.3.32
trame
synonyme critiqué de DLPDU

3.3.33
horloge grand-mère
dans un domaine, horloge constituant la source de temps primaire pour la synchronisation d'horloge à l'aide du protocole PTP

[SOURCE: CEI 61588:2009, 3.1.13]

3.3.34
instance
occurrence physique permettant d'identifier un objet parmi d'autres au sein d'une même classe d'objets

EXEMPLE "California" (La Californie) est une instance de la classe d'objets "state" (état).

Note 1 à l'article: Les termes objet, instance et instance d'objet font référence à une instance particulière.

3.3.35
attribut d'instance
attribut qui est réservé à l'usage exclusif d'une instance d'objet et n'est pas commun à la classe d'objets

3.3.36
instancié
se dit d'un objet créé dans un appareil

3.3.37
interopérabilité
capacité des entités de couche Utilisateur à accomplir des opérations coordonnées et coopératives en utilisant les services de la FAL

3.3.38
Keeper (Gardien)
objet chargé de distribuer les données de configuration d'une liaison à tous les nœuds de la liaison

3.3.39
little endian (petit-boutiste)
modèle d'organisation de la mémoire qui stocke l'octet de poids le plus faible à l'adresse la plus basse ou, pour le transfert, qui transfère en premier l'octet d'ordre le plus faible

Note 1 à l'article: Les types de données Type 2 natif sont envoyés dans un ordre little endian.

3.3.40
Lpacket
Link packet
élément d'information d'application qui contient une taille, un octet de contrôle, une étiquette et des données de liaison

Note 1 à l'article: Les couches Liaison de données des homologues utilisent des Lpacket pour envoyer et recevoir des unités de données de service provenant des couches supérieures de la pile OSI.

3.3.41
informations de gestion
informations accessibles via le réseau qui facilitent la gestion de l'exploitation du système de bus de terrain, y compris la couche application

Note 1 à l'article: La gestion inclut des fonctions telles que la commande, la surveillance et le diagnostic.

3.3.42
horloge mère
dans le cas d'un chemin de communication PTP unique, horloge constituant la source de temps à laquelle toutes les autres horloges de ce chemin se synchronisent

[SOURCE: CEI 61588:2009, 3.1.17]

3.3.43
membre
élément d'un attribut qui est structuré comme une partie d'une matrice

3.3.44
Message Router (routeur de message)
objet au sein d'un nœud qui distribue les demandes de messagerie vers les objets d'application appropriés

3.3.45
connexion multipoint
connexion d'un nœud à plusieurs autres nœuds

Note 1 à l'article: Les connexions multipoints permettent que les messages issus d'un seul producteur soient reçus par plusieurs nœuds consommateurs.

3.3.46
réseau
ensemble de nœuds reliés par un support de communication d'un type ou d'un autre, avec d'éventuels répétiteurs intermédiaires, ponts, routeurs et passerelles de couche inférieure

3.3.47
objet
représentation abstraite d'un composant particulier dans un appareil; il s'agit généralement d'un ensemble de données (sous la forme de variables) et de méthodes (procédures) associées, destiné à l'exploitation des données dont l'interface et le comportement sont clairement définis

3.3.48
service spécifique à un objet
service réservé à l'usage exclusif de la classe d'objets qui le définit

3.3.49
horloge ordinaire
horloge possédant un seul port PTP dans un même domaine, qui maintient l'échelle de temps utilisée dans ce domaine

Note 1 à l'article: Une horloge ordinaire peut jouer le rôle de source de temps, c'est-à-dire être une horloge mère, ou se synchroniser à une autre horloge, c'est-à-dire être une horloge esclave.

[SOURCE: CEI 61588:2009, 3.1.22, modifiée – deuxième phrase transformée en Note]

3.3.50
émetteur
client chargé d'établir un chemin de connexion vers une cible

3.3.51
horloge parente
horloge mère à laquelle une horloge est synchronisée

[SOURCE: CEI 61588:2009, 3.1.23]

3.3.52
connexion point à point
connexion établie précisément entre deux objets d'application

3.3.53
Precision Time Protocol (protocole de temps de précision)
PTP
protocole défini par la CEI 61588:2009

Note 1 à l'article: Utilisé comme adjectif, ce terme indique que le nom modifié est spécifié ou interprété dans le contexte de la CEI 61588:2009.

[SOURCE: CEI 61588:2009, 3.1.28, modifiée– deuxième phrase transformée en Note]

3.3.54
produire
acte d'envoyer des données destinées à être reçues par un consommateur

3.3.55
producteur
nœud chargé de l'envoi des données

3.3.56
message PTP
un des types de message définis dans la CEI 61588:2009

[SOURCE: CEI 61588:2009, 3.1.33]

3.3.57
port PTP
point d'accès logique d'une horloge pour les communications PTP, au réseau de communications

[SOURCE: CEI 61588:2009, 3.1.35]

3.3.58
récepteur
utilisateur de service qui reçoit une primitive confirmée ou non confirmée, ou fournisseur de service qui reçoit une APDU confirmée ou non confirmée

3.3.59
ressource
capacité de traitement ou d'information d'un sous-système

3.3.60
expéditeur
utilisateur de service qui envoie une primitive confirmée ou non confirmée, ou fournisseur de service qui envoie une APDU confirmée ou non confirmée

3.3.61
serveur
a) rôle d'un AREP (point d'extrémité de relation d'applications) dans lequel il retourne au client qui a lancé la demande une APDU confirmée de réponse de service
b) objet qui fournit des services à un autre objet (client)

3.3.62
service
opération ou fonction qu'un objet et/ou une classe d'objets réalise ou assure à la demande d'un autre objet et/ou d'une autre classe d'objets

3.3.63
horloges synchronisées
(avec une incertitude spécifiée) deux horloges qui ont la même époque et pour lesquelles les mesures de la durée d'un événement unique à une heure arbitraire ne diffèrent pas de plus que l'incertitude spécifiée

[SOURCE: CEI 61588:2009, modification de 3.1.40 (reformulé)]

3.3.64
System Time (heure système)
temps absolu tel que défini par la synchronisation de CPF2 dans le contexte d'un système de temps distribué où tous les appareils ont une horloge locale qui est synchronisée avec une horloge mère commune

Note 1 à l'article: Dans le contexte de CPF2, System Time (heure système) est une valeur entière de 64 bits exprimée en unités de nanosecondes avec une valeur 0 correspondant à la date 1970-01-01 (c'est-à-dire 1er janvier 1970).

3.3.65
cible
nœud d'extrémité auquel une connexion est établie

3.3.66
nœud temporaire
nœud transitoire

3.3.67
transaction ID (ID de transaction)
champ dans un en-tête UCMM qui met une réponse en correspondance avec la demande associée

3.3.68
horloge transparente
appareil qui mesure le temps nécessaire à un message d'événement PTP pour traverser l'appareil et qui fournit cette information aux horloges recevant ce message d'événement PTP

[SOURCE: CEI 61588:2009, modification de 3.1.46 (tronqué)]

3.3.69
gestionnaire de message non connecté
UCMM
composant au sein d'un nœud qui émet et reçoit des messages explicites non connectés, et les envoie directement à l'objet Message Router (routeur de message)

3.3.70
service non connecté
service de messagerie qui ne repose pas sur l'établissement d'une connexion entre les appareils avant d'autoriser des échanges d'informations

3.3.71
vendor ID (ID de vendeur)
identification de chaque fabricant/vendeur de produit par un numéro unique

Note 1 à l'article: Les identificateurs de vendeur sont attribués par ODVA, Inc.

3.4 Abréviations et symboles

ASCII	American Standard Code for Information Interchange
CID	ID de connexion (Connection ID)
DLL	Couche Liaison de données (Data-Link Layer)
DSCP	Point de code de service différencié (DiffServ CodePoint)
IGMP	Protocole de gestion de groupe Internet (Internet Group Management Protocol; voir IETF RFC 1112 et IETF RFC 2236)
IP	Protocole Internet (Internet Protocol; voir IETF RFC 791)
IPv4	Protocole Internet version 4 (voir IETF RFC 791)
IPv6	Protocole Internet version 6 (voir IETF RFC 791)

OSI	Interconnexion des systèmes ouverts (Open Systems Interconnection; voir ISO/CEI 7498-1)
PDU	unité de données de protocole (Protocol Data Unit)
PHB	Comportement par saut (Per-Hop Behavior)
PTP	Protocole de temps de précision (Precision Time Protocol voir CEI 61588:2009)
QoS	Qualité de service (Quality of Service)
Rcv	Recevoir
Rx	Recevoir
SDU	unité de données de service (Service Data Unit)
SEM	Matrice d'événements d'états (State Event Matrix)
STD	Diagramme de transitions d'états (State Transition Diagram), utilisé pour décrire le comportement d'objet
TCP	Protocole de contrôle de terminal (Terminal Control Protocol; voir IETF RFC 793)
ToS	Type de service (Type of Service)
TPDU	Unité de données de protocole de transport (Transport Protocol Data Unit)
Tx	Emettre
UDP	Protocole de datagramme d'utilisateur (User Datagram Protocol; voir IETF RFC 768)
Xmit	Emettre
CM_API	Intervalle réel entre paquets (Actual Packet Interval)
O2T ou O⇒T	Emetteur vers cible (Originator to Target; paramètres de connexion)
CM_RPI	Intervalle demandé entre paquets (Requested Packet Interval)
TUI	Identificateur unique de table (Table Unique Identifier)
T2O ou T⇒O	Cible vers émetteur (Target to Originator; paramètres de connexion)

3.5 Conventions

3.5.1 Concept général

La couche FAL se compose d'un ensemble d'ASE orientés objet. Chaque ASE est spécifié dans un paragraphe distinct. Chaque spécification d'ASE est constituée de trois parties: ses définitions de classe, ses services et sa spécification de protocole. Les deux premiers éléments sont contenus dans la CEI 61158-5-2. La spécification des protocoles pour chacun des éléments ASE est définie dans la présente norme.

Les définitions de classe définissent les attributs des classes prises en charge par chaque ASE. Les attributs sont accessibles à partir des instances de la classe qui utilise les services ASE de gestion spécifiés dans la CEI 61158-5-2. La spécification de service définit les services fournis par l'élément ASE.

La présente norme emploie les conventions de description énoncées dans l'ISO/CEI 10731.

La police gras est utilisée dans la présente norme pour mettre en valeur des noms de paramètre ou des éléments importants dans le texte. On utilise aussi les italiques dans les Tableaux et les Notes pour assurer une meilleure visibilité.

3.5.2 Spécification d'attribut

Les attributs sont définis dans un Tableau d'attributs utilisant le format et les termes définis à la Figure 1.

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs

Figure 1 – Format et termes du tableau d'attributs

Attribute ID doit être une valeur d'identification entière unique attribuée à un attribut. Les plages valides pour les ID d'attribut doivent être celles spécifiées en 4.1.10.1.3. L'ID d'attribut doit identifier l'attribut particulier auquel on accède.

Name doit se référer au nom attribué à l'attribut. Un attribut peut contenir des sous-éléments, qui peuvent aussi avoir des noms, comme c'est le cas avec le type de données **STRUCT of**. Le nom d'attribut doit être le nom qui apparaît le premier ou à la première ligne de la colonne de nom (la ligne qui contient l'ID d'attribut).

Chaque attribut doit se voir attribuer un **Data type** qui doit être soit un type de données élémentaire, soit un type de données dérivé. Les types de données spécifiés pour les attributs définis doivent être utilisés dans toutes les mises en œuvre.

NOTE Les types de données élémentaires sont définis dans la CEI 61158-5-2.

Semantics of values doit spécifier la signification de la ou des valeurs de l'attribut. Si cette information a besoin de plus d'espace que le tableau ne peut lui en fournir, elle sera placée immédiatement après le Tableau d'attributs de classe. Une référence adéquate à cette information sera incluse dans le Tableau d'attributs de classe.

Si un Attribut de classe est facultatif, une valeur par défaut ou une méthode de traitement de cas spéciaux doit être définie afin que le Client (Demandeur) puisse traiter le message d'erreur qui se produit lors de l'accès aux objets qui choisissent de ne pas mettre en œuvre l'attribut de classe.

3.5.3 Services communs

3.5.3.1 Définitions des Service_PDU

Chaque service a des paramètres uniques pour la demande et la réponse. Les paramètres de demande et de réponse de service doivent être définis avec l'aide des tableaux de paramètres Request/Response (c'est-à-dire: Demande/Réponse) de service tels que définis à la Figure 2.

Nom	Type	Sémantique des valeurs

Figure 2 – Paramètre de demande/réponse de service

Name doit faire référence au nom donné au paramètre de demande/réponse de service.

Type doit spécifier le type de données du paramètre de demande/réponse de service.

Semantics of values doit spécifier la signification des valeurs du paramètre de demande/réponse de service (par exemple, "la valeur est en comptes en microsecondes").

3.5.3.2 Réponse Get_Attribute_All

3.5.3.2.1 Définition générale

Lorsque le service commun Get_Attribute_All est inclus dans la liste des services System/Object Management (Gestion de système/d'objets) pris en charge, la réponse **Get_Attribute_All** doit être détaillée pour cette classe: la séquence ou l'ordre des données retournées dans la Service_ResponsePDU doit être spécifié(e). Trois méthodes sont admises pour spécifier la Get_Attribute_All Service_ResponsePDU:

- énumérer l'ordonnancement des attributs dans le message de réponse;
- spécifier la matrice de données réelle du message de réponse;

- combiner les deux méthodes ci-dessus afin que la matrice de données réelle contienne également les numéros d'attribut.

Quelle que soit la méthode utilisée, les règles spécifiées dans le Tableau 1 doivent être respectées pour spécifier la Service_ResponsePDU d'une classe d'objets tant pour les attributs de classe que pour les attributs d'instance.

Tableau 1 – Règles relatives au service de réponse Get_Attribute_All

Numéro de règle	Règle
1	Si la définition de la réponse Get_Attribute_All inclut des attributs facultatifs, les valeurs par défaut doivent être spécifiées dans la description de la réponse. Les attributs facultatifs qui se trouvent en fin de liste et ne sont pas mis en œuvre peuvent être omis dans les données de réponse. Les attributs facultatifs qui se trouvent en milieu de liste et ne sont pas mis en œuvre doivent faire partie des données de réponse et des valeurs par défaut doivent leur être attribuées. Si l'un quelconque des attributs facultatifs suivants est inclus dans une spécification d'objet, mais n'est pas pris en charge dans la mise en œuvre de l'objet, les conditions suivantes doivent être satisfaites: – si l'attribut de classe "Optional attribute list" (Liste d'attributs facultatifs) n'est pas pris en charge, la valeur par défaut zéro doit être insérée dans la matrice de réponse et aucun numéro d'attribut facultatif ne doit suivre; – si l'attribut de classe "Optional service list" (Liste de services facultatifs) n'est pas pris en charge, la valeur par défaut zéro doit être insérée dans la matrice de réponse et aucun numéro de service facultatif ne doit suivre.
2	Si de nouveaux attributs sont ajoutés à un objet existant, les attributs en question doivent être ajoutés à la fin de la liste d'attributs de réponse ou matrice de données afin d'assurer la compatibilité avec les différentes révisions de l'objet.
3	Quelle que soit la méthode utilisée pour spécifier la réponse, cette spécification doit être faite de façon non ambiguë, en incluant les règles pour traiter les champs de longueur variable et le bourrage.
4	Pour les objets spécifiés dans la présente norme, la réponse Get_Attribute_All ne doit comprendre que les attributs ouverts; elle ne doit pas comprendre d'attributs spécifiques à un vendeur.

Le Tableau 2 donne un exemple de méthode d'ordonnancement des attributs permettant de spécifier la partie "données de service" d'une réponse Get_Attribute_All pour des attributs de niveau classe relatifs à un objet qui prend en charge les attributs de classe **récupérables facultatifs** 1, 2, 3 et 4.

Tableau 2 – Exemples de données de réponse spécifiques à un service/objet de niveau classe pour Get_Attribute_All

ID d'attribut de classe	Nom d'attribut et valeur par défaut
1	Revision (Révision), valeur par défaut = 0x0001
2	Max Instance (Instance max.), valeur par défaut = 0x0000
3	Number of Instances (Nombre d'instances), valeur par défaut = 0x0000
4	Attribute list (Liste d'attributs), nombre d'attributs, valeur par défaut = 0x0000

Le Tableau 3 donne un exemple de méthode de matrice de données permettant de spécifier la partie "données de service" d'une réponse Get_Attribute_All pour des attributs de niveau classe relatifs à un objet qui prend en charge les attributs de classe **récupérables facultatifs** 1, 2, 3 et 4.

Tableau 3 – Exemple de méthode de matrice de données Get_Attribute_All

Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Revision (octet de poids faible) Valeur par défaut = 1							
1	Revision (octet de poids fort) Valeur par défaut = 0							
2	Max Instance (octet de poids faible) Valeur par défaut = 0							
3	Max Instance (octet de poids fort) Valeur par défaut = 0							
4	Number of Instances (octet de poids faible) Valeur par défaut = 0							
5	Number of Instances (octet de poids fort) Valeur par défaut = 0							
6	Optional Attribute List: nombre d'attributs (octet de poids faible) Valeur par défaut = 0							
7	Optional Attribute List: nombre d'attributs (octet de poids fort) Valeur par défaut = 0							
8	Optional Attribute List: attribut optionnel n° 1 (octet de poids faible)							
9	Optional Attribute List: attribut optionnel n° 1 (octet de poids fort)							
2n + 6	Optional Attribute List: attribut optionnel n° n (octet de poids faible)							
2n + 7	Optional Attribute List: attribut optionnel n° n (octet de poids fort)							

3.5.3.2.2 Révisions

La réponse Get_Attribute_All définie pour un objet peut voir sa taille augmenter après chaque révision de l'objet; cependant, afin d'assurer l'interopérabilité, le format de la première partie de la réponse doit être analysable par un client de la révision plus ancienne de l'objet. Les Clients (Demandeurs) peuvent ne pas utiliser cette exigence de compatibilité.

NOTE L'attribut de classe Revision n'est pas la révision d'une mise en œuvre (qui est reflétée dans l'objet Identity, bits de statut Minor/Major revision (Révision mineure/majeure)), mais la révision de la class definition (définition de classe).

3.5.3.3 Demande Set_Attribute_All

3.5.3.3.1 Définition générale

Lorsque le service commun Set_Attribute_All est inclus dans la liste des services communs pris en charge, le format de la demande **Set_Attribute_All** doit être inclus dans la spécification de l'objet. La séquence ou l'ordre des données fournies dans la partie "Service Data" (Données de service) de la demande doit être spécifié(e). Trois méthodes sont admises pour spécifier la demande Set_Attribute_All:

- énumérer l'ordre des attributs dans le message de demande;
- spécifier la matrice de données réelle du message de demande;
- combiner les deux méthodes ci-dessus afin que la matrice de données réelle contienne également les numéros d'attribut.

Quelle que soit la méthode utilisée, les règles spécifiées dans le Tableau 4 doivent être respectées pour spécifier la demande Set_Attribute_All d'un objet dans la spécification de l'objet tant pour les attributs de classe que pour les attributs d'instance.

Tableau 4 – Règles relatives au service de demande Set_Attribute_All

Numéro de règle	Règle
1	Un objet doit prendre en charge le service Set_Attribute_All seulement si tous les attributs réglables montrés dans la demande Set_Attribute_All sont mis en œuvre comme étant réglables.
2	Les valeurs par défaut doivent être spécifiées pour tous les attributs facultatifs qui ne sont pas mis en œuvre. Si une mise en œuvre ne prend pas en charge un attribut facultatif, elle doit accepter la valeur par défaut pour l'attribut non pris en charge.
3	Si de nouveaux attributs réglables sont ajoutés à un objet existant, les attributs en question doivent être ajoutés à la fin de la liste d'attributs de demande ou matrice de données.
4	Quelle que soit la méthode utilisée pour spécifier la réponse, cette spécification doit être faite de façon non ambiguë, en incluant les règles pour traiter les champs de longueur variable et le bourrage.
5	La réponse à la demande Set_Attribute_All pour les objets spécifiés dans la présente norme doit uniquement comprendre les attributs ouverts. Elle ne doit pas inclure d'attributs spécifiques à un vendeur.

Le Tableau 5 donne un exemple de méthode d'ordonnancement des attributs permettant de spécifier la partie "données de service" d'une demande Set_Attribute_All pour des attributs de niveau instance relatifs à un objet qui prend en charge les attributs d'instance réglables exigés 7, 8, 9, 10, 11 et 12.

Tableau 5 – Exemple de méthode d'ordonnancement d'attributs Set_Attribute_All

ID d'attribut de classe	Nom d'attribut
7	Output Range
8	Value Data Type
9	Fault State
10	Idle State
11	Fault Value
12	Idle Value

Le Tableau 6 donne un exemple de méthode de matrice de données permettant de spécifier la partie "données de service" d'une demande Set_Attribute_All pour des attributs de niveau instance relatifs à un objet qui prend en charge les attributs d'instance réglables exigés 7, 8, 9, 10, 11 et 12.

Tableau 6 – Exemple de méthode de matrice de données Set_Attribute_All

Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Output Range							
1	Value Data Type							
2	Fault State							
3	Idle State							
4	Fault Value (octet de poids faible)							
5	Fault Value (octet de poids fort)							
6	Idle Value (octet de poids faible)							
7	Idle Value (octet de poids fort)							

3.5.3.3.2 Révisions

Un serveur traitant une demande Set_Attribute_All peut avoir besoin de traiter plus de données qu'il n'en est prévu par le serveur si le client a mis en œuvre une plus récente

révision de cet objet alors que le serveur a une version plus ancienne. Il en résulte que l'objet serveur peut avoir à traiter une demande Set_Attribute_All qui contient plus de données parce que les attributs supplémentaires n'avaient pas été reconnus. Si le serveur reçoit plus de données dans une demande Set_Attribute_All qu'il n'en attend, le serveur doit répondre par un code de statut général égal à 0x15 (Too much data (Trop de données)).

NOTE L'attribut de classe Revision n'est pas la révision d'une mise en œuvre (qui est reflétée dans l'objet Identity, bits de statut Minor/Major revision (Révision mineure/majeure)), mais la révision de la class definition (définition de classe).

3.5.4 Conventions relatives aux diagrammes d'états

3.5.4.1 Généralités

Les changements d'état peuvent être déclenchés par des événements (internes ou externes) ou par des invocations de services. La réaction aux invocations de services peut dépendre de la ou des valeurs du ou des attributs accessibles au service.

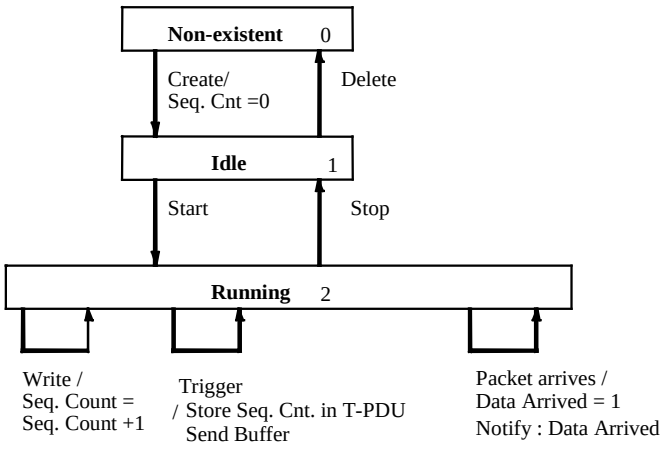
Le comportement du diagramme d'états doit être défini pour la combinaison de:

- l'**événement** que le diagramme reçoit;
- l'**état** dans lequel se trouve le diagramme lorsqu'il reçoit la notification d'un événement modificateur d'état.

Pour définir le comportement en ces termes, un diagramme de transitions d'états (State Transition Diagram, STD) et une matrice d'événements d'états (State Event Matrix, SEM) sont utilisés lorsque cela est applicable dans la spécification du diagramme d'états.

3.5.4.2 Diagramme de transitions d'états (STD)

Un **événement** est un stimulus externe qui peut entraîner une transition d'état. Un STD illustre graphiquement les états d'un objet et inclut des événements, des appels de services et des changements d'attributs qui le font passer à un autre état. La Figure 3 montre un exemple de STD.



Légende

Anglais	Français
Non-existent	Inexistant
Create/	Créer/
Seq. Cnt = 0	Compte de séquences = 0
Delete	Supprimer

Anglais	Français
Idle	Au repos
Start	Démarrer
Stop	Arrêter
Running	En marche
Write /	Ecrire/
Seq. Count =	Compte de séquences =
Seq. Count +1	Compte de séquences +1
Trigger	Déclencher
/ Store Seq. Cnt. in T-PDU	/Stocker compte de séquences dans T-PDU
Send Buffer	Envoyer tampon
Packet arrives /	Paquet arrive/
Data Arrived = 1	Données arrivées = 1
Notify: Data Arrived	Notifier: Données arrivées

Figure 3 – Exemple de STD

NOTE L'événement qui est la cause principale de la transition d'état est suivi d'une barre "/". La notification à la couche supérieure est repérée par "Notify" (Notifier):

3.5.4.3 Matrice d'événements d'états

Un **état** est le mode de fonctionnement actuellement actif de l'objet diagramme d'états (par exemple, Running (En marche) ou Idle (Au repos)).

Une matrice d'événements d'états est un tableau qui énumère tous les événements, services et modifications possibles dans les valeurs d'attribut qui déclenchent un changement d'état. Elle indique également la réponse de l'objet à l'événement en fonction de l'état de l'objet lorsqu'il reçoit notification de l'événement en question.

Le format d'une matrice d'événements d'états est tel que montré dans le Tableau 7.

Tableau 7 – Format d'une matrice d'événements d'états

Evénement	Etat		
	Etat 1	....	Etat n
Description de l'événement A	Fonction déclenchée par l'événement A dans l'Etat 1 (le cas échéant) Notification à l'utilisateur de FAL (le cas échéant) Transition vers un autre état (le cas échéant)		Fonction déclenchée par l'événement A dans l'Etat n (le cas échéant) Notification à l'utilisateur de FAL (le cas échéant) Transition vers un autre état (le cas échéant)
.....	.....		.....
Description de l'événement X	Fonction déclenchée par l'événement X dans l'Etat 1 (le cas échéant) Notification à l'utilisateur de FAL (le cas échéant) Transition vers un autre état (le cas échéant)		Fonction déclenchée par l'événement X dans l'Etat n (le cas échéant) Notification à l'utilisateur de FAL (le cas échéant) Transition vers un autre état (le cas échéant)
En l'absence de fonction, de notification ou de transition, l'entrée correspondante ne doit pas être incluse dans le tableau.			

3.5.4.4 Exemple de matrice d'événements d'états

Le Tableau 8 montre un exemple de diagramme d'états comportant trois états:

- **Non-existent** (Inexistant): l'objet n'a pas encore été créé; les objets passent à l'état Existent (Existant) via le service Create (Créer) (si l'objet peut être créé de manière dynamique) ou à la mise sous tension (si l'objet est fixé par conception/mise en œuvre);
- **Idle** (Au repos): l'objet accepte des services (par exemple, Get_Attribute_Single), mais ne produit pas de données vers la liaison ou n'en consomme pas en provenance de la liaison;
- **Running** (En marche): l'objet accomplit toutes ses fonctions spécifiées.

Tableau 8 – Exemple de matrice d'événements d'états

Evénement	Etat		
	Non-existent (Inexistant)	Inactif	Running (En marche)
Créer	Passer à Idle	Erreur: L'objet existe déjà.	Erreur: L'objet existe déjà.
Supprimer	Erreur: L'objet n'existe pas. (Code d'erreur général 0x16)	Passer à Non-existent	Erreur: Object State Conflict (Conflit d'état d'objet) (Code d'erreur général 0x0C)
Démarrer	Erreur: L'objet n'existe pas. (Code d'erreur général 0x16)	Passer à Running	Erreur: Object State Conflict (Conflit d'état d'objet) (Code d'erreur général 0x0C)
Arrêter	Erreur: L'objet n'existe pas. (Code d'erreur général 0x16)	Erreur: Object State Conflict (Conflit d'état d'objet) (Code d'erreur général 0x0C)	Passer à Idle
Get_Attribute_Single	Erreur: L'objet n'existe pas. (Code d'erreur général 0x16)	Valider/gérer la demande. Retourner la réponse	Valider/gérer la demande. Retourner la réponse
Set_Attribute_Single	Erreur: L'objet n'existe pas. (Code d'erreur général 0x16)	Valider/gérer la demande. Retourner la réponse	Erreur: Object State Conflict (Conflit d'état d'objet) (Code d'erreur général 0x0C)

4 Syntaxe abstraite

4.1 Syntaxe abstraite des PDU de la FAL

4.1.1 Généralités

La FAL_PDU doit être soit une UCMM_PDU, soit une Transport_PDU.

UCMM_PDU sert à acheminer de l'information pour les services sans connexion.

Transport_PDU sert à acheminer de l'information pour les services orientés connexion.

Un message connecté présume des ressources et des paramètres négociés au préalable à sa source, à sa/ses destination(s) et en tous les éventuels points de transit intermédiaires. Ces ressources sont référencées par un identificateur de connexion unique et peuvent ne pas être contenues dans chaque message. Seul l'ID de connexion est indispensable pour identifier le message et faire référence à ses paramètres connexes, permettant ainsi des économies considérables en termes de rendement de message.

Un message non connecté donne le moyen de communiquer sur la liaison locale sans ressources préalablement négociées en la destination et il doit donc transporter les détails complets de l'ID de la destination, les descripteurs de données internes et les détails complets de l'ID de la source si une réponse est demandée. Les messages non connectés sont surtout utilisés pour créer des connexions.

Le service non connecté est fourni par le gestionnaire de messages non connectés (UCMM "Unconnected Message Manager"). Les messages reçus par le biais de l'UCMM sont transmis au routeur de messages (MR "Message Router"), qui les dirige pour exécution vers l'objet interne adéquat. Les connexions sont établies par des messages non connectés spécifiques envoyés par le biais du Connection Manager (CM, gestionnaire de connexion) à l'aide des services de l'UCMM. Les connexions peuvent être établies soit vers le Message Router (à des fins de messagerie), soit directement vers un objet d'application. La cible de la connexion est spécifiée à l'aide d'un chemin de connexion et peut être située sur une liaison locale ou distante, en passant par plusieurs nœuds routeurs intermédiaires. Une fois qu'une connexion est établie avec un objet d'application, l'UCMM, le MR et le CM ne sont plus indispensables, car les données seront échangées directement avec l'objet connecté, en fonction de l'ID de connexion correspondant. Les messages connectés envoyés au Message Router seront transmis pour exécution à l'objet interne adéquat.

4.1.2 Structure des PDU

4.1.2.1 Structure de UCMM_PDU

UCMM_PDU est une séquence d'un UCMM_Header, suivi soit de OM_Service (ASE Object Management), soit de CM_Service (ASE Connection Manager).

OM_Service doit être soit OM_Request, soit OM_Response.

OM_Request doit comprendre MR_Request_Header et une Service_RequestPDU.

Service_RequestPDU doit être l'une des suivantes:

- Get_Attribute_Single_RequestPDU
- Get_Attribute_All_RequestPDU
- Get_Attribute_List_RequestPDU
- Set_Attribute_Single_RequestPDU
- Set_Attribute_All_RequestPDU
- Set_Attribute_List_RequestPDU
- Reset_RequestPDU
- Create_RequestPDU
- Delete_RequestPDU
- Start_RequestPDU
- Stop_RequestPDU
- Find_Next_Object_Instance_RequestPDU
- NOP_RequestPDU
- Apply_Attributes_RequestPDU
- Save_RequestPDU
- Restore_RequestPDU
- Get_Member_RequestPDU
- Set_Member_RequestPDU
- Insert_Member_RequestPDU

- Remove_Member_RequestPDU
- Group_Sync_RequestPDU
- Add_AckData_Path_RequestPDU
- Remove_AckData_Path_RequestPDU
- Get_Enum_String_RequestPDU
- Symbolic_Translation_RequestPDU
- Connection_Bind_RequestPDU
- Producing_Application_Lookup_RequestPDU

OM_Response doit comprendre MR_Response_Header et une Service_ResponsePDU.

Service_ResponsePDU doit être l'une des suivantes:

- Get_Attribute_Single_ResponsePDU
- Get_Attribute_All_ResponsePDU
- Get_Attribute_List_ResponsePDU
- Set_Attribute_Single_ResponsePDU
- Set_Attribute_All_ResponsePDU
- Set_Attribute_List_ResponsePDU
- Reset_ResponsePDU
- Create_ResponsePDU
- Delete_ResponsePDU
- Start_ResponsePDU
- Stop_ResponsePDU
- Find_Next_Object_Instance_ResponsePDU
- NOP_ResponsePDU
- Apply_Attributes_ResponsePDU
- Save_ResponsePDU
- Restore_ResponsePDU
- Get_Member_ResponsePDU
- Set_Member_ResponsePDU
- Insert_Member_ResponsePDU
- Remove_Member_ResponsePDU
- Group_Sync_ResponsePDU
- Add_AckData_Path_ResponsePDU
- Remove_AckData_Path_ResponsePDU
- Get_Enum_String_ResponsePDU
- Symbolic_Translation_ResponsePDU
- Connection_Bind_ResponsePDU
- Producing_Application_Lookup_ResponsePDU

CM_Service doit être soit CM_Request, soit CM_Response.

CM_Request doit comprendre MR_Request_Header et une CM_RequestPDU.

CM_RequestPDU doit être l'une des suivantes:

- Forward_Open_RequestPDU
- Forward_Close_RequestPDU
- Large_Forward_Open_RequestPDU
- Unconnected_Send_RequestPDU
- Get_Connection_Data_RequestPDU
- Search_Connection_Data_RequestPDU
- Get_Connection_Owner_RequestPDU

CM_Response comprend MR_Response_Header et une CM_ResponsePDU.

CM_ResponsePDU doit être l'une des suivantes:

- Forward_Open_ResponsePDU
- Forward_Close_ResponsePDU
- Large_Forward_Open_ResponsePDU
- Unconnected_Send_ResponsePDU
- Get_Connection_Data_ResponsePDU
- Search_Connection_Data_ResponsePDU
- Get_Connection_Owner_ResponsePDU

4.1.2.2 Structure de Transport_PDU

Transport_PDU est une séquence d'un Transport_Header, suivi soit de OM_Service (ASE Object Management), soit de Application Data (Données d'application).

OM_Service est défini en 4.1.2.1 ci-dessus.

Les données d'application proviennent de la source à laquelle la connexion a été faite.

4.1.3 UCMM_PDU

Le Paragraphe 4.1.3 définit les exigences des PDU d'UCMM quand une couche Liaison de données de Type 2 est utilisée.

Toutes les UCMM_PDU doivent être envoyées et reçues en utilisant l'étiquette fixe 0x83 ou l'étiquette de gestion 0x88. Lorsqu'un appareil est alimenté en énergie, l'UCMM doit invoquer le service `DLL_enable_fixed_request` de la couche Liaison de données pour demander que les Lpackets à étiquette fixe 0x83 ou 0x88 soient acheminés vers l'UCMM. Tous les expéditeurs et récepteurs de TPDU doivent respectivement utiliser les primitives de service `DLL_fixed_request.req` et `DLL_fixed_request.ind` de la couche Liaison de données. Le paramètre paquet de ces services doit être préfixé avec l'en-tête suivant pour former la PDU Transport avant l'envoi vers la couche Liaison de données. Le format de l'en-tête de l'UCMM_ PDU est montré dans le Tableau 9.

Tableau 9 – Format de l'en-tête de l'UCMM_PDU

Nom de paramètre	Format
command_code	USINT
Timeout	USINT
TransactionID	UINT

Le paramètre command_code doit spécifier le type de l'UCMM_PDU comme montré dans le Tableau 10. Les paquets UCMM reçus avec un command_code réservé doivent être rejetés sans acquittement.

Tableau 10 – Codes de commande de l'UCMM

Command_code	Description
0	Réservé
1	acquitter une demande
2	demande avec répétition de tentative jusqu'à acquittement
3	réponse avec répétition de tentative jusqu'à acquittement
4	demande sans acquittement et sans réponse
5	acquitter une réponse
6	réponse sans répétition de tentative (pas d'acquittement)
7	demande avec répétition de tentative jusqu'à la réponse (pas d'acquittement)
8	Demande sans répétition de tentative qui doit donner lieu à une réponse de code 6
9 à 255	Réservé

Le champ timeout (expiration de délai) doit spécifier la durée de la transaction dans un format de virgule flottante de 8 bits. Les 5 bits de poids fort du champ doivent être un exposant non signé qui n'est pas biaisé. Les 3 bits de poids faible doivent être les 3 bits de poids le plus faible d'une mantisse non signée de 4 bits. Le bit de poids le plus fort de la mantisse doit être 1 et ne doit pas apparaître de façon explicite dans la représentation en 8 bits. La virgule binaire doit être positionnée entre le 1 implicite et le reste de la mantisse. Les unités de la durée de transaction calculée avec l'aide du champ timeout doivent être des millisecondes.

NOTE Si on applique les règles de priorité de l'ANSI C, le nombre de tops d'horloge de 0,125 ms est (8 | temporisation & 7) << (temporisation >> 3 & 31).

Le champ transactionID doit être composé de deux sous-champs:

- les 10 bits de poids faible, RECORD, doivent identifier une transaction spécifique;
- les 6 bits de poids fort, SEQUENCE, doivent être utilisés pour la détection de doublons. Il doit être incrémenté pour chaque message unique figurant sur ce RECORD; cependant, il ne doit pas être incrémenté sur une répétition de tentative.

Un seul message doit être en cours pour un RECORD donné.

Si plusieurs messages en cours sont requis, plusieurs RECORD sont requis.

Lorsqu'un paquet à étiquette fixe "I'm alive" est reçu d'un nœud (voir la CEI 61158-4-2, 6.10 et 8.1), toutes les transactions de l'UCMM avec le nœud en question doivent être abandonnées.

4.1.4 Transport_Header

4.1.4.1 Généralités

Le contenu des en-têtes Transport varie en fonction de la classe de transport sélectionnée au cours de l'établissement de la connexion.

4.1.4.2 Classe 0 (vide ou de base)

L'en-tête de transport de classe 0 doit être un nombre non signé de 16 bits. Cet en-tête doit être écrit dans le tampon de TPDU par le transport et doit être simplement rejeté, par le transport, du paquet entrant en provenance du consommateur. Le Tableau 11 montre le format de l'en-tête.

Tableau 11 – En-tête de transport de classe 0

Nom de paramètre	Format
don't_care	UINT

4.1.4.3 Classe 1 (détection de doublons)

L'en-tête de transport de classe 1 doit être un compte de séquences non signé de 16 bits. Cet en-tête doit être écrit dans le tampon de TPDU par le transport et doit être lu par le transport dans le paquet entrant en provenance du consommateur. Le Tableau 12 montre le format de l'en-tête.

Tableau 12 – En-tête de transport de classe 1

Nom de paramètre	Format
sequence_count	UINT

4.1.4.4 Classe 2 (acquitté)

Comme l'en-tête de transport de classe 1, l'en-tête de transport de classe 2 doit être un compte de séquences de 16 bits. Cet en-tête doit être écrit dans le tampon de TPDU par le transport et doit être lu par le transport dans le paquet entrant en provenance du consommateur. Le Tableau 13 montre le format de l'en-tête.

Tableau 13 – En-tête de transport de classe 2

Nom de paramètre	Format
sequence_count	UINT

4.1.4.5 Classe 3 (vérifié)

Comme l'en-tête de transport de classe 1, l'en-tête de transport de classe 3 doit être un compte de séquences de 16 bits. Cet en-tête doit être écrit dans le tampon de TPDU par le transport et doit être lu par le transport dans le paquet entrant en provenance du consommateur. Le Tableau 14 montre le format de l'en-tête.

Tableau 14 – En-tête de transport de classe 3

Nom de paramètre	Format
sequence_count	UINT

4.1.4.6 Classe 4 (non bloquante), classe 5 (non bloquante, fragmentante) et classe 6 (connexion multipoint, fragmentante)

NOTE Le contenu du paragraphe de l'édition précédente est devenu obsolète.

4.1.4.7 Format de données Listen only O⇒T

La connexion O=>T doit utiliser un format heartbeat spécifique (absence d'en-tête 32 bits, 0 octet de données d'application).

4.1.4.8 Format de données Input only O⇒T

La connexion O=>T doit utiliser un format heartbeat spécifique (absence d'en-tête 32 bits, 0 octet de données d'application).

4.1.4.9 Format de données Exclusive owner O⇒T

Les connexions de propriétaire exclusif doivent avoir l'un des deux formats de transfert temps réel:

- en-tête de 32 bits, taille fixe;
- absence d'en-tête, taille variable.

L'en-tête de 32 bits préfixé aux données temps réel doit avoir la forme montrée dans le Tableau 15.

Tableau 15 – En-tête de données temps réel – propriétaire exclusif

Paramètre	Format	Taille (bits)
Run_idle	RUN_IDLE	1
Réservé	UDINT	31

Le fanion `run_idle` (bit 0) doit être mis (`1 = RUN`, EN MARCHE) pour indiquer que les données suivantes doivent être envoyées à l'application cible. Il doit être effacé (`0 = IDLE`, AU REPOS) pour indiquer que l'événement repos doit être envoyé à l'application cible. Le champ `reserved` (bits 1 à 31) doit être réservé et mis à 0.

Si le format de transfert no_header est utilisé, la réception d'un paquet au-delà de l'en-tête de transport doit indiquer le mode RUN. Si le paquet est tronqué après l'en-tête de transport, la cible doit être un événement de repos.

4.1.4.10 Format de données Redundant owner O⇒T

4.1.4.10.1 Format général

Le propriétaire redondant doit avoir un en-tête de 32 bits préfixé aux données temps réel. Cet en-tête doit avoir la forme montrée dans le Tableau 16.

Tableau 16 – En-tête de données temps réel – propriétaire redondant

Paramètre	Format	Taille (bits)
Run_idle	RUN_IDLE	1
COO	BOOLEAN	1
ROO	USINT	2
Réservé	UDINT	28

4.1.4.10.2 Fanion Run_idle

Le fanion run_idle (bit 0) doit avoir la même signification que dans la connexion de propriétaire exclusif et doit être soit RUN, soit IDLE. Le champ reserved (bits 4 à 31) doit être réservé et mis à 0.

4.1.4.10.3 Fanion Claim output ownership (COO)

Le fanion COO doit être mis (1) lorsqu'une application d'émetteur souhaite que sa connexion soit la connexion propriétaire de l'application cible. Le fanion COO doit être réinitialisé (0)

lorsqu'une application d'émetteur ne souhaite pas que sa connexion soit la connexion propriétaire de l'application cible. Lorsque la connexion propriétaire réinitialise (0) son fanion COO, ses connexions enfants doivent être contrôlées pour détecter la présence d'un fanion COO mis (1). Le nouveau propriétaire doit être l'une quelconque des connexions dont le fanion COO est mis.

NOTE Il en résulte un comportement indéterminé si plusieurs connexions ont également leur fanion COO mis.

4.1.4.10.4 Valeur de priorité Ready for ownership of outputs (ROO)

La valeur de priorité ROO doit être non nulle lorsqu'une application d'émetteur ne souhaite pas forcer sa connexion à être la connexion propriétaire de l'application cible, mais est prête à être la connexion propriétaire s'il n'y a pas d'application d'émetteur prétendant être la connexion propriétaire. La valeur de priorité ROO doit être zéro lorsque l'application d'émetteur ne souhaite pas être la connexion propriétaire de la connexion cible, et n'est pas prévue être la connexion propriétaire s'il n'y a pas d'application d'émetteur prétendant être la connexion propriétaire. La valeur de priorité ROO doit être utilisée seulement lorsque le fanion COO est réinitialisé.

La valeur du champ ROO peut aller de 0 à 3. Chacune des applications de l'émetteur doit déterminer une valeur unique ROO différente de zéro.

4.1.4.10.5 Détermination de la connexion propriétaire

Les applications d'émetteur doivent déterminer laquelle parmi elles a la connexion propriétaire. La connexion propriétaire doit être déterminée par l'application d'émetteur qui met son fanion COO. Dans les situations dans lesquelles plusieurs applications d'émetteur ont leur fanion COO mis ou dans lesquelles aucune connexion d'émetteur n'a son fanion mis, les règles ci-après doivent être appliquées par le transport cible pour déterminer la connexion propriétaire.

- Il ne doit y avoir aucune connexion propriétaire tant qu'une application d'émetteur n'a pas envoyé un paquet temps réel avec le fanion COO mis.
- S'il n'y a qu'une seule application d'émetteur qui a eu le fanion COO mis dans son dernier paquet temps réel, l'application d'émetteur en question doit avoir la connexion propriétaire.
- S'il y a plusieurs applications d'émetteur qui ont eu le fanion COO mis dans leur dernier paquet temps réel, la dernière application d'émetteur qui a fait passer son fanion COO de l'état réinitialisé à l'état mis doit avoir la connexion propriétaire.
- Si l'application d'émetteur avec la connexion propriétaire réinitialise son fanion COO, ferme sa connexion ou si la connexion en question expire, et qu'aucune autre application d'émetteur n'a son fanion mis, l'application d'émetteur ayant la priorité ROO non nulle la plus élevée doit avoir la connexion propriétaire.
- Si toutes les applications d'émetteur ont leurs fanions COO réinitialisés et des valeurs de priorité ROO mises à zéro, il ne doit pas y avoir de connexion propriétaire.
- Lorsque le premier paquet temps réel contenant un fanion COO mis est reçu par le transport cible, l'application d'émetteur qui a envoyé le paquet temps réel doit avoir la connexion propriétaire.

4.1.4.10.6 Transport d'événements et de données vers une application cible

Les événements liés à la connexion issus de chacune des connexions de propriétaires redondants doivent être combinés à l'aide des règles ci-après de manière que l'application cible ne voie qu'une seule connexion de propriétaire exclusif:

- Tant qu'aucune connexion propriétaire n'a été déterminée initialement, le transport ne doit pas indiquer de réception de données temps réel à l'application cible.
- Si une connexion propriétaire est déterminée, le transport doit indiquer à l'application cible l'événement compatible avec les données temps réel des connexions propriétaires. Si le fanion Run/Idle est réinitialisé dans les données temps réel pour la connexion propriétaire,

le transport doit indiquer l'état Idle à l'application cible. Si le fanion Run/Idle est mis dans les données temps réel pour la connexion propriétaire, le transport doit indiquer l'état Run et les données temps réel à l'application cible.

- Si une connexion propriétaire a été déterminée au préalable, mais qu'aucun émetteur n'en réclame la propriété ni n'est prêt pour la propriété, le transport doit indiquer l'état Idle à l'application cible.
- Si toutes les connexions redondantes sont fermées ou ont été interrompues par temporisation, le transport cible doit indiquer à l'application cible l'événement compatible avec la dernière connexion à avoir été fermée ou interrompue par temporisation.

4.1.5 CM_PDU

4.1.5.1 Généralités

Les services d'établissement et de maintenance de connexion sont fournis par Connection Manager. Ce sont

CM_Forward_Open (correspond aux PDU Forward_Open et Large_Forward_Open)

CM_Forward_Close (correspond aux PDU Forward_Close)

CM_Unconnected_Send (correspond aux PDU Unconnected_Send)

CM_Get_Connection_Data (correspond aux PDU Get_Connection_Data)

CM_Search_Connection_Data (correspond aux PDU Search_Connection_Data)

CM_Get_Connection_Owner (correspond aux PDU Get_Object_ Owner)

4.1.5.2 Connection Manager

L'objet Connection Manager sert à gérer l'établissement et la maintenance des connexions de communication.

Les objets Connection Manager situés aux différents nœuds doivent communiquer à l'aide des services de l'UCMM décrits dans la CEI 61158-5-2. Les TPDU avec lesquelles communiquent les Connection Manager d'homologues doivent être dérivées des services du Connection Manager.

4.1.5.3 Forward_Open

4.1.5.3.1 Généralités

Le service Forward_Open sert à établir une connexion avec un appareil cible. Ce service a pour résultat qu'une connexion locale est établie sur chaque liaison sur le chemin.

NOTE Le traitement d'un seul service Forward_Open peut donner lieu à la création de plusieurs instances actives de l'objet Connection.

Un Forward Open peut soit être un Forward_Open non nul soit un Forward_Open nul. Un Forward_Open non nul est une demande de service Forward_Open pour laquelle au moins un des types de connexion dans le champ O2T_ ou T2O_connection_parameters n'est pas 00 (NULL). Un Forward_Open nul est une demande de service Forward_Open pour laquelle le type de connexion des deux champs O2T_ et T2O_connection_parameters est 00 (NULL), dans ce cas aucune connexion n'est établie.

Un Forward_Open (soit nul soit non nul) peut soit concorder soit ne pas concorder. Il y a demande de service Forward_Open concordante reçue par l'appareil cible quand Connection Triad concorde avec une connexion existante.

Le Paragraphe 4.1.5.3.2 décrit et la liste ci-dessous résume l'utilisation des combinaisons de Forward_Open non nul / nul et concordant / non concordant.

- non nul / non concordant – ouvre une connexion;

- non nul / concordant – erreur;
- nul / non concordant – envoie un ping ou configure un appareil;
- nul / concordant – reconfigure.

La réponse à la Forward_Open_Request_PDU est une Forward_Open_ResponsePDU. La réponse Forward_Open doit contenir le numéro de série de connexion de l'émetteur, le numéro de série et de vendeur du propriétaire et les ID de connexion (CID) nécessaires pour parachever la connexion si succès et des informations d'erreur si échec de la connexion.

4.1.5.3.2 Variantes Forward_Open

4.1.5.3.2.1 Forward_Open non nul, non concordant

La demande Forward_Open établit les connexions réseau, de transport et d'application. Une connexion d'application consiste en une seule connexion de transport et une ou deux connexions réseau qui sont constituées elles-mêmes de plusieurs connexions de liaison. Chaque segment de port dans le chemin de connexion utilise une connexion de liaison. Le service Forward_Open entre deux appareils construit une ou deux connexions de liaison telles que spécifiées par le paramètre de connexion réseau et les intervalles demandés entre paquets (CM_RPI). Sachant qu'il peut être requis jusqu'à deux connexions réseau pour une seule connexion de transport, celles-ci sont différenciées par les désignations O2T et T2O; O2T signifie "originator to target" (émetteur vers cible) et T2O "target to originator" (cible vers émetteur).

Le chemin spécifié dans MR_Request_Header pour le service Forward_Open ne doit contenir aucun segment de clé électronique (Electronic Key). Des segments de clé électronique peuvent être compris dans le chemin spécifié au paramètre connection_path du service Forward_Open.

Success (Succès) doit être retourné lorsque la connexion demandée a été établie à partir de ce point dans le chemin. Cette réponse doit aussi indiquer le numéro de série de la connexion et l'intervalle réel entre paquets de la connexion. Une fois qu'une réponse réussie a été reçue, la connexion doit être ouverte à partir de ce point dans le chemin. Les cibles doivent attendre au moins 10 secondes après l'envoi de `CM_Forward_Open_Good_Response` pour le premier paquet sur une connexion.

4.1.5.3.2.2 Forward_Open non nul, concordant

Un Forward_Open non nul pour lequel Connection Triad correspond à une connexion existante doit retourner un statut général = 0x01, statut étendu = 0x0100 (connexion en utilisation ou Duplicate Forward_Open).

Il convient que le comportement d'émetteur dans le cas Forward_Open non nul concordant soit de fermer et rétablir la connexion ou bien d'attendre l'expiration de la connexion et la rétablir. Il doit être reconnu que, dans le second cas, la connexion doit mettre 60 s (première expiration de données) pour expirer avant de pouvoir être rétablie.

4.1.5.3.2.3 Forward_Open nul, non concordant

On peut utiliser un Forward_Open nul pour lequel Connection Triad ne correspond à aucun paramètre de connexion existante pour les fonctions suivantes.

a) Envoyer un ping à un appareil, les caractéristiques étant les suivantes:
   - le chemin unique d'application "20 01 24 01" (c'est-à-dire objet Identity);
   - un segment de clé électronique peut être inclus;
   - ne comprend aucun segment de données;
   - aucune connexion n'est établie.

b) Configurer une application d'appareil, les caractéristiques étant les suivantes:

- un chemin d'application de configuration et un segment de données doivent faire partie de la demande (la demande est envoyée à l'application spécifiée par le chemin et elle est appliquée);
- si toute la configuration ne peut pas être appliquée, aucune des configurations ne doit être appliquée et le code d'erreur approprié doit être retourné;
- un segment de clé électronique peut être inclus;
- aucune connexion n'est établie.

Un appareil cible doit répondre à une demande Forward_Open nulle par l'une des erreurs suivantes quand la cible ne prend pas en charge la fonction demandée ("envoyer un ping à un appareil" ou "configurer une application d'appareil"):

- La fonction Forward_Open nulle n'est pas prise en charge (statut général 0x01, statut étendu 0x0132) (recommandé).
- Paramètre de connexion réseau invalide (statut général 0x01, statut étendu 0x0108) (déconseillé).

4.1.5.3.2.4 Forward_Open nul, concordant

Un Forward_Open nul pour lequel Connection Triad correspond à des paramètres d'une connexion existante peut être utilisé pour reconfigurer une application d'appareil cible. Les nœuds intermédiaires doivent toujours transmettre ces demandes. Les comportements associés sont les suivants:

- un segment de données et un chemin d'application de configuration doivent faire partie de la demande et être envoyés à l'application pour modifier la configuration de l'application;
- si la configuration complète ne peut être appliquée, aucun élément de la configuration ne doit être appliqué et un code d'erreur approprié doit être retourné;
- la connexion ne doit pas être interrompue à cause de cette demande.

Un appareil cible doit répondre à une demande Forward_Open nulle par l'une des erreurs suivantes quand la cible ne prend pas en charge la fonction demandée ("reconfigurer une application de l'appareil cible"):

- La fonction Forward_Open nulle n'est pas prise en charge (statut général 0x01, statut étendu 0x0132) (recommandé).
- Paramètre de connexion réseau invalide (statut général 0x01, statut étendu 0x0108) (déconseillé).

Si l'interprétation des données consommées / produites change à cause de l'opération de reconfiguration, il faut faire attention aux applications qui consomment et produisent. Il n'y aucun mécanisme spécifié chargé de la coordination entre les applications qui consomment et celles qui produisent, de sorte qu'un changement dans la signification des données en temps réel peut être à l'origine d'un fonctionnement imprévu. Les appareils disposent de la possibilité de rejeter une demande de reconfiguration, une erreur d'état d'objet (statut général = 0x0c) empêchant cette situation.

4.1.5.3.3 Format de la demande Forward_Open

Le format de la TPDU de demande Forward_Open doit avoir la forme montrée dans le Tableau 17.

Tableau 17 – Format de la demande Forward_Open

Nom de paramètre	Format
priority_and_tick	SWORD

Nom de paramètre	Format
connection time-out ticks	USINT
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
connection timeout multiplier	USINT
reserved[3]	USINT
O2T_CM_RPI	UDINT
O2T_connection_parameters [a]	UINT
T2O_CM_RPI	UDINT
T2O_connection_parameters [a]	UINT
xport_type_and_trigger	SWORD
connection_path_size	USINT
connection_path	Padded EPATH
a Connection type de ce paramètre doit être 00 (NULL) pour un Forward_Open nul.	

4.1.5.3.4 Format de la réponse Forward_Open si succès

Si le paramètre `status` de la `CM_open_response` est zéro (absence d'erreur), le format de la TPDU de réponse Forward_Open doit avoir la forme montrée dans le Tableau 18.

Tableau 18 – Format de la réponse Forward_Open_Good

Nom de paramètre	Format
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
O2T_CM_API	UDINT
T2O_CM_API	UDINT
Application_reply_size; (en mots de 16 bits)	USINT
Réservé	USINT
application_reply[]	UINT

Si un objet Connection est instancié pour prendre en charge cette connexion:

- les valeurs de O2T_CM_API et de O2T_CM_API sont utilisées pour l'attribut Expected_packet_rate de l'objet Connection, après conversion en millisecondes.
- Expected_packet_rate ne sera pas utilisé pour l'objet Connection Inactivity/Watchdog Timer. Voir 4.1.6.2 pour traiter la temporisation de la connexion.

4.1.5.3.5 Format de la réponse Forward_Open si échec

Si le paramètre `status` de la `CM_open_response` est différent de zéro (erreur), le format de la TPDU de réponse Forward_Open doit avoir la forme montrée dans le Tableau 19.

Tableau 19 – Format de la réponse Forward_Open_Bad

Nom de paramètre	Format
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
remaining_path_size; (en mots de 16 bits)	USINT
Réservé	USINT

Ce format doit être utilisé pour toutes les défaillances du système de connexions. La connexion demandée ne doit pas être établie et les mots de statut spécifiques à l'objet doivent contenir des informations relatives à la cause de la défaillance. Le paramètre `remaining_path_size` doit contenir la longueur du chemin au point où la connexion a failli. Cette information peut être utilisée pour déboguer le problème.

Dans la réponse d'échec, le paramètre restant `remaining_path_size` doit avoir une taille "pre-stripped" (précouchée). Il doit s'agir de la taille du chemin lorsque le nœud reçoit d'abord la demande et n'a pas encore commencé à la traiter. Un nœud cible peut retourner soit la taille "pre-stripped" (précouchée), soit 0 pour le `remaining_path_size` restant.

Un doublon de service Forward_Open doit être défini comme un service Forward_Open dont les `originator_vendor_ID`, `connection_serial_number` et `originator_serial_number` correspondent aux paramètres d'une connexion existante. Si le doublon de service Forward_Open est un service Forward_Open vide (défini comme le type de connexion dans lequel les champs de paramètres de connexion réseau tant O2T que T2O sont NULL), le service Forward_Open doit être transmis à l'application pour un traitement ultérieur. Les demandes Forward_Open vides ("null") peuvent être utilisées pour reconfigurer la connexion. Le Connection Manager dans les nœuds intermédiaires peut ne pas allouer de ressources supplémentaires pour un doublon de demande Forward_Open, car les ressources ont déjà été allouées. Si le doublon de demande Forward_Open n'est pas NULL, un statut général = 0x01, statut étendu = 0x0100, doit être retourné.

4.1.5.4 Large_Forward_Open

4.1.5.4.1 Généralités

Large_Forward_Open doit avoir la même fonction et le même comportement que Forward_Open, excepté qu'il doit permettre l'établissement de connexions supérieures à 511 octets.

La réponse à la Large_Forward_Open_Request_PDU est une Large_Forward_Open_ResponsePDU. Toutes les autres exigences sont identiques à celles spécifiées pour le service Forward_Open en 4.1.5.3.

4.1.5.4.2 Format de la demande Large_Forward_Open

Le format de la TPDU de demande Large_Forward_Open doit avoir la forme montrée dans le Tableau 20.

Tableau 20 – Format de la demande Large_Forward_Open

Nom de paramètre	Format
priority_and_tick	SWORD
connection time-out ticks	USINT
O2T_CID	UDINT
T2O_CID	UDINT

Nom de paramètre	Format
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
connection timeout multiplier	USINT
reserved[3]	USINT
O2T_CM_RPI	UDINT
O2T_ex_connection_size	UINT
O2T_connection_parameters	UDINT
T2O_CM_RPI	UDINT
T2O_ex_connection_size	UINT
T2O_connection_parameters	UDINT
xport_type_and_trigger	SWORD
connection_path_size	USINT
connection_path	Padded EPATH

Le champ taille de la connexion dans les deux connection_parameters doit être réservé dans ce cas et mis à zéro. Les paramètres ex_connection_size doivent être utilisés à la place pour spécifier la taille maximale de la connexion (jusqu'à 65 535).

4.1.5.4.3 Format de la réponse Large_Forward_Open si succès

Si le paramètre `status` de la `CM_open_response` est zéro (absence d'erreur), le format de la TPDU de réponse Large_Forward_Open doit avoir la forme montrée dans le Tableau 21.

Tableau 21 – Format de la réponse Large_Forward_Open_Good

Nom de paramètre	Format
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
O2T_CM_API	UDINT
T2O_CM_API	UDINT
Application_reply_size; (en mots de 16 bits)	USINT
Réservé	USINT
application_reply[]	UINT

4.1.5.4.4 Format de la réponse Large_Forward_Open si échec

Si le paramètre `status` de la `CM_open_response` est différent de zéro (erreur), le format de la TPDU de réponse Large_Forward_Open doit avoir la forme montrée dans le Tableau 22.

Tableau 22 – Format de la réponse Large_Forward_Open_Bad

Nom de paramètre	Format
connection_serial_number	UINT
originator_vendor_ID	UINT

Nom de paramètre	Format
originator_serial_number	UDINT
remaining_path_size; (en mots de 16 bits)	USINT
Réservé	USINT

Soit un appareil cible, soit un routeur intermédiaire doit retourner un code d'erreur étendu Invalid Connection Size, avec le mot de statut supplémentaire de la taille maximale de connexion prise en charge, si la taille de la connexion demandée n'est pas prise en charge lors du traitement de la demande Large_Forward_Open.

4.1.5.5 Forward_Close

4.1.5.5.1 Généralités

La demande Forward_Close doit retirer une connexion de tous les nœuds participant à la connexion d'origine. La demande Forward_Close doit être envoyée entre les Connection Manager comme spécifié dans le connection_path. La demande Forward_Close doit entraîner la désaffectation de toutes les ressources dans tous les nœuds participant à la connexion, y compris les ID de connexion, le temps d'émission de liaison et les tampons des mémoires internes.

Si un nœud intermédiaire ne peut pas trouver la connexion qui est à fermer (elle peut avoir expiré au nœud), la demande Forward_Close doit encore être transmise aux nœuds aval ou à l'application cible (par le truchement de `CM_close_indication`).

NOTE Forward_Close est toujours transmis pour permettre à des nœuds aval ou à l'application cible de libérer les éventuelles ressources qui étaient allouées pour la connexion.

4.1.5.5.2 Format de la demande Forward_Close

Le format de demande Forward_Close doit avoir la forme montrée dans le Tableau 23.

Le chemin spécifié dans MR_Request_Header pour le service Forward_Close ne doit contenir aucun segment de clé électronique (Electronic Key). Les segments de clé électronique peuvent faire partie du chemin spécifié dans le paramètre connection_path du service Forward_Close.

Tableau 23 – Format de la demande Forward_Close

Nom de paramètre	Format
connection_priority/tick time	SWORD
connection_timeout	USINT
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
connection_path_size	USINT
Réservé	USINT
connection_path	Padded EPATH

4.1.5.5.3 Format de la réponse Forward_Close si succès

La réponse Forward_Close doit être envoyée entre les Connection Manager en réponse à une demande Forward_Close et doit contenir le statut de la demande de fermeture. Si le paramètre `status` de la `CM_close_response` est zéro (absence d'erreur), le format de la TPDU de réponse Forward_Close doit avoir la forme montrée dans le Tableau 24.

Tableau 24 – Format de la réponse Forward_Close_Good

Nom de paramètre	Format
connection serial number	UINT
originator_vendor ID	UINT
originator serial number	UDINT
application_reply_size; en mots de 16 bits	USINT
Réservé	USINT
application_reply[]	UINT

Une réponse Forward_Close réussie doit être retournée lorsque la fermeture a été acquittée par tous les nœuds à partir de ce point du chemin jusqu'à la fin du chemin. La réponse doit indiquer les `connection_serial_number`, `originator_vendor_ID` et `originator_serial_number` de la connexion fermée. Une fois qu'une réponse indiquant le succès a été reçue, la connexion doit être fermée à partir de ce point du chemin jusqu'à la fin. L'application cible peut, facultativement, renvoyer des informations spécifiques à l'application dans la réponse de fermeture réussie. Si aucune information `application_reply` n'est contenue dans le service, le paramètre `application_reply_size` doit être zéro.

4.1.5.5.4 Format de la réponse Forward_Close si échec

Si le paramètre `status` de la `CM_close_response` est différent de zéro (erreur), le format de la réponse Forward_Close doit avoir la forme montrée dans le Tableau 25.

Tableau 25 – Format de la réponse Forward_Close_Bad

Nom de paramètre	Format
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
remaining_path_size	USINT
réservé	USINT

Les mots de statut spécifiques à l'objet doivent contenir les informations relatives à la cause de l'échec. Le paramètre `remaining_path_size` doit contenir la taille du chemin à partir du point auquel la connexion de fermeture a failli.

Dans la réponse d'échec, le paramètre `remaining_path_size` doit avoir une taille "pre-stripped" (précouchée). Il doit s'agir de la taille du chemin lorsque le nœud reçoit d'abord la demande et n'a pas encore commencé à la traiter. Un nœud cible doit retourner soit la taille "pre-stripped" (précouchée), soit 0 pour le `remaining_path_size`.

4.1.5.6 Unconnected_Send

4.1.5.6.1 Généralités

Le service Unconnected_Send doit permettre à une application d'envoyer un message à un appareil sans établir une connexion d'abord. Le service Unconnected_Send doit utiliser l'objet Connection Manager en chaque nœud intermédiaire pour transmettre le message et se rappeler le chemin de retour. L'UCMM de chaque liaison doit être utilisé pour transmettre la demande d'un Connection Manager à un autre tout comme dans le cas du service Forward_Open; toutefois, aucune connexion ne doit être bâtie. Le service Unconnected_Send doit être envoyé au Connection Manager local et doit être envoyé entre les nœuds intermédiaires. Lorsqu'un nœud intermédiaire supprime le dernier segment de port, la demande intégrée de message doit être formatée comme un OM_Request et envoyée au port et à l'adresse de liaison du dernier segment de port à l'aide de l'UCMM pour ce type de liaison.

NOTE Le nœud cible ne voit jamais le service Unconnected_Send, mais seulement une requête de message intégrée arrivant par l'intermédiaire de l'UCMM.

4.1.5.6.2 Format de la demande Unconnected_Send

Le format de demande Unconnected_Send doit avoir la forme montrée dans le Tableau 26.

Tableau 26 – Format de la demande Unconnected_Send

Nom de paramètre	Format	Description
priority/tick time	SWORD	Utilisé pour calculer les informations de temporisation de la demande
conn time-out ticks	USINT	
Message_size	UINT	Nombre d'octets dans le PDU Message Router intégré
PDU Message Router	OM_Request	PDU Message Router intégré
pad	USINT	N'existe que si Message_size est une valeur impaire Cela permet aux champs restants d'être alignés sur une frontière de 16 bits.
route_path_size	USINT	Nombre de mots de 16 bits dans le champ route_path
Réservé	USINT	Réservés, doivent être mis à zéro (0).
route_path	Padded EPATH	Indique l'itinéraire vers l'appareil cible distant

Le chemin spécifié dans MR_Request_Header pour le service Unconnected_Send ne doit contenir aucun segment de clé électronique (Electronic Key). Il est admis que des segments de clé électronique se trouvent dans le chemin spécifié dans le MR_Request_Header de la demande intégrée du message et dans le paramètre route_path du service Unconnected_Send.

Le chemin de l'itinéraire doit avoir la forme suivante:

[segment1 Electronic Key] Segment1 de port	} Port Segment Group pour le 1er nœud intermédiaire }
[[segment2 Electronic Key] Segment2 de port]	} Port Segment Group pour le 2e nœud intermédiaire }
...	
[[segmentN Electronic Key] SegmentN de port]	} Port Segment Group pour le Ne nœud intermédiaire }

Tous les segments entre crochets [...] sont facultatifs. Si le segment Electronic Key est présent, la cible doit évaluer la clé. Voir 4.1.9.3 pour la définition du segment Port et 4.1.9.4.2 pour la définition du segment Electronic Key.

4.1.5.6.3 Réponse Unconnected_Send

La réponse Unconnected_Send doit être générée par le dernier nœud intermédiaire à partir de la réponse de l'UCMM générée par le nœud cible ou par un nœud intermédiaire à la suite d'une expiration d'UCMM, d'un problème avec le message incorporé ou d'un problème avec la demande de service non connectée (Unconnected Service Request) elle-même. Le paquet doit être acheminé d'un nœud intermédiaire à l'autre en utilisant les informations stockées lorsque la demande Unconnected_Send était traitée. La réponse doit contenir un en-tête avec des informations de statut relatives à la demande et une réponse de longueur variable générée par le nœud cible.

4.1.5.6.4 Format de la réponse Unconnected_Send si succès

Ce format doit être utilisé à la réception d'une réponse Unconnected_Send réussie. La structure de la réponse est montrée dans le Tableau 27.

Tableau 27 – Format de la réponse Unconnected_Send_Good

Nom de paramètre	Format
application_response [ ]	OCTET

Le champ application_response contient les données que retourne l'appareil / l'objet cible dans le Service_ResponsePDU pour la demande intégrée. Si le Service_ResponsePDU que retourne l'appareil / l'objet cible ne contenait pas de données, ce champ doit être vide.

EXEMPLE Ce champ contiendrait Attribute Data en réponse à une demande Get_Attribute_Single intégrée.

4.1.5.6.5 Format de la réponse Unconnected_Send si échec

Si le service Unconnected_Send échoue, le code Service retourné dans MR_Response_Header pour le service Unconnected_Send peut correspondre soit au code Service envoyé dans les données de service Unconnected_Send (demande intégrée de message) soit au code Unconnected_Send Service lui-même:

- quand un routeur détecte l'erreur, le code service de la réponse Unconnected_Send (0xD2) doit être retourné dans MR_Response_Header;
- quand l'appareil cible détecte l'erreur, le code du Service de réponse pour la demande de message intégrée doit être retourné dans MR_Response_Header;

EXEMPLE si l'appareil cible détecte l'erreur et que le service demandé était un Get_Attribute_Single, le code de service dans la MR_Response_Header sera 0x8E.

Cela représente un avantage pour les émetteurs et les routeurs parce que:

- parmi les deux codes de service (Unconnected Send et intégré), le code de service qui a échoué est retourné à l'émetteur;
- les routeurs ne sont pas obligés d'analyser le code de service du message intégré dans la demande Unconnected_Send.

Le format du service de réponse Unconnected_Send pour une défaillance doit avoir la forme montrée dans le Tableau 28.

Tableau 28 – Format de la réponse Unconnected_Send_Bad

Nom de paramètre	Format
remaining_path_size	USINT
pad	USINT

Le champ remaining_path_size n'existe qu'en cas d'erreur de type d'itinéraire, il indique le nombre de mots dans le chemin originel de l'itinéraire (paramètre route_path de la demande Unconnected Send) vu par le routeur qui détecte l'erreur. Le champ de bourrage est réservé et doit avoir pour valeur 0.

4.1.5.6.6 Modifications de CP 2/3

4.1.5.6.6.1 Généralités

Les services de Connection Manager ne sont pas pris en charge par les nœuds CP 2/3 quand ils sont la cible de la demande. Il en résulte que seuls les nœuds qui indiquent l'itinéraire en provenance ou à destination de CP 2/3 prennent en charge l'objet Connection Manager. Quand le nœud cible du message se trouve en CP 2/3, ces routeurs sont tenus de traduire le message en format de message explicite normal CP 2/3 (voir la CEI 62026-3). Il en résulte que les nœuds CP 2/3 ne demandent pas de modification pour accepter les messages transmis depuis d'autres sous-réseaux de Type 2.

Pour permettre à ces routeurs de traiter les nombreuses transactions en cours sur CP 2/3, le service Unconnected_Send de Connection Manager est modifié quand il traverse un sous-réseau CP 2/3, comme le spécifie 4.1.5.6.6.2 et 4.1.5.6.6.3.

4.1.5.6.6.2 Modification de la demande Unconnected_Send

Quand les données de service Unconnected_Send sont envoyées à un sous-réseau CP 2/3, elles sont préfixées par un identificateur de transaction de 16 bits. Cet identificateur de transaction est produit par l'appareil demandeur et il est retourné par le routeur avec la réponse provenant de la cible. Le Tableau 29 représente la demande modifiée de service.

Tableau 29 – Format de la demande Unconnected_Send (modifiée)

Nom de paramètre	Format	Description
Transaction_ID	UINT	Utilisé pour la gestion des transactions dans l'appareil demandeur CP 2/3 et les routeurs CP 2/3. Ce champ sert aux émetteurs de message pour faire concorder les transactions sur CP 2/3 non transmis aux autres sous-réseaux.
priority/tick time	SWORD	Utilisé pour calculer les informations de temporisation de la demande
conn time-out ticks	USINT	
Message_size	UINT	Nombre d'octets dans le PDU Message Router intégré
PDU Message Router	OM_Request	PDU Message Router intégré
pad	USINT	N'existe que si Message_size est une valeur impaire Cela permet aux champs restants d'être alignés sur une frontière de 16 bits.
route_path_size	USINT	Nombre de mots de 16 bits dans le champ route_path
Réservé	USINT	Réservés, doivent être mis à zéro (0).
route_path	Padded EPATH	Indique l'itinéraire vers l'appareil cible distant

4.1.5.6.6.3 Modification de la réponse Unconnected_Send

Un routeur CP 2/3 doit toujours retourner une réponse réussie à une demande de service Unconnected_Send. Cette réponse réussie peut en fait indiquer qu'une erreur a eu lieu. Cela est nécessaire parce que des informations supplémentaires d'erreur sont indispensables pour traiter les erreurs d'itinéraire et que ces informations supplémentaires ne satisfont pas au format réponse d'erreur (Error Response) défini pour CP 2/3 (voir la CEI 62026-3). Comme l'indique 4.1.5.6.5, le code Service retourné peut soit être le code Service envoyé à l'intérieur des données de service Unconnected_Send soit le code de service Unconnected_Send. Le code de service réponse d'erreur 0x94 (voir la CEI 62026-3) n'est jamais retourné.

Les réponses de service réussies et non réussies figurent dans le Tableau 30 et dans le Tableau 31. Ce sont les données qui sont dans la zone des données de service application_response; le code de service réponse se trouve plus en avant dans le paquet.

Tableau 30 – Format de la réponse Unconnected_Send_Good (modifié)

Nom de paramètre	Format	Description
Transaction_ID	UINT	Renvoie en écho la valeur reçue dans la demande Unconnected_Send associée.
Etat général	USINT	Cette valeur est zéro (0) pour les transactions réussies.
Réservé	USINT	Doit être zéro (0).
actual_application_response [ ]	OCTET	

Le champ actual_application_response contient les données retournées par l'appareil / l'objet cible dans le Service_ResponsePDU pour la demande intégrée. Si le Service_ResponsePDU que retourne l'appareil / l'objet cible ne contenait pas de données, ce champ doit être vide.

EXEMPLE Ce champ contiendrait Attribute Data en réponse à une demande intégrée.

Cette réponse Service Data associée à une réponse non réussie Unconnected_Send est définie ci-dessous.

Tableau 31 – Format de la réponse Unconnected_Send_Bad (modifié)

Nom de paramètre	Format	Description
Transaction_ID	UINT	Renvoie en écho la valeur reçue dans la demande Unconnected_Send associée.
Statut général	USINT	Un des codes Statut général qui figure dans le Tableau 180. S'il y a eu une erreur d'itinéraire, elle doit se limiter aux valeurs détectées par les appareils routeurs telles qu'elles figurent au Tableau 179.
Taille du statut supplémentaire (Additional Status)	USINT	Nombre de mots de 16 bits dans la matrice Additional Status
Additional Status [ ]	UINT	Quand il retourne une erreur provenant d'une cible qui est un nœud CP 2/3, le statut supplémentaire doit contenir l'Additional Error Code de 8 bits provenant de la cible dans les 8 bits au poids le plus faible et un zéro (0) dans les 8 bits au poids le plus élevé.
remaining_path_size	USINT	Ce champ n'existe qu'en cas d'erreur de type d'itinéraire, il indique le nombre de mots dans l'itinéraire d'origine (paramètre route_path de la demande Unconnected_Send) comme le voit le routeur qui détecte l'erreur.

4.1.5.7 Get_Connection_Data

4.1.5.7.1 Généralités

Ce service doit retourner les paramètres associés à un numéro de connexion spécifié. Le connection number peut être différent d'un appareil à l'autre même pour la même connexion. Le connection number correspond au décalage dans l'attribut de Connection Manager qui énumère le statut des connexions.

NOTE Ce service peut être utilisé pour les diagnostics de réseaux.

4.1.5.7.2 Format de la demande Get_Connection_Data

Le format de la demande Get_Connection_Data doit avoir la forme montrée dans le Tableau 32.

Tableau 32 – Format de la demande Get_Connection_Data

Nom de paramètre	Format
connection_number	UINT

4.1.5.7.3 Format de la réponse Get_Connection_Data

Le format de la TPDU de réponse Get_Connection_Data doit avoir la forme montrée dans le Tableau 33.

Tableau 33 – Format de la réponse Get_Connection_Data

Nom de paramètre	Format
connection_number	UINT
connection_state	UINT
originator_port	UINT
target_port	UINT
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
originator_O2T_CID	UDINT
target_O2T _CID	UDINT
O2T_connection timeout multiplier	USINT
reserved1[3]	USINT
originator_O2T_CM_RPI	UDINT
originator_O2T_CM_API	UDINT
originator_T2O_CID	UDINT
target_ T2O_CID	UDINT
T2O_connection timeout multiplier	USINT
reserved2[3]	USINT
originator_T2O_CM_RPI	UDINT
originator_T2O_CM_API	UDINT

4.1.5.8 Search_Connection_Data

4.1.5.8.1 Généralités

Ce service doit retourner les paramètres associés à une connexion spécifiée au sein d'un appareil identifié par vendeur et numéro de série.

NOTE Ce service peut être utilisé pour les diagnostics de réseaux.

4.1.5.8.2 Format de la demande Search _Connection_Data

Le format de la demande Search_Connection_Data doit avoir la forme montrée dans le Tableau 34.

Tableau 34 – Format de la demande Search_Connection_Data

Nom de paramètre	Format
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT

4.1.5.8.3 Format de la réponse Search _Connection_Data

Le format de la réponse Search_Connection_Data doit être le même que celui de la réponse issue du service Get_Connection_Data (voir 4.1.5.7).

4.1.5.9 Get_Connection_Owner

4.1.5.9.1 Généralités

Le service Get_Connection_Owner du Connection Manager doit retourner des données concernant la/les connexion(s) qui possède(nt) un objet particulier. Il doit être mis en œuvre dans tout appareil qui accepte des connexions redondantes.

4.1.5.9.2 Format de la demande Get_Connection_Owner

Le format de la TPDU de demande Get_Connection_Owner doit avoir la forme montrée dans le Tableau 35.

Tableau 35 – Format de la demande Get_Connection_Owner

Nom de paramètre	Format
réservé	USINT
path_size	USINT
path[]	Padded EPATH

Le champ réservé doit être mis à zéro.

4.1.5.9.3 Format de la réponse Get_Connection_Owner

Le format de la TPDU de réponse Get_Connection_Owner doit avoir la forme montrée dans le Tableau 36.

Tableau 36 – Format de la réponse Get_Connection_Owner

Nom de paramètre	Format
number_of_connections	USINT
number_claiming_ownership	USINT
number_ready_for_ownership	USINT
last_action	USINT
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT

4.1.6 Composants de PDU CM

4.1.6.1 Paramètres de connexion réseau

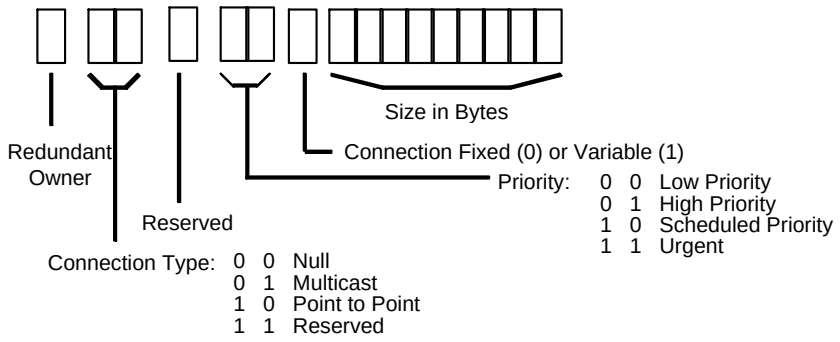
4.1.6.1.1 Format

Les paramètres de connexion réseau doivent être fournis sous la forme d'un seul mot de 16 bits qui contient cinq champs, comme montré à la Figure 4. Les champs au sein du mot de 16 bits doivent indiquer

- le type de connexion réseau souhaitée;
- si la taille de tampon est variable ou fixe;
- la priorité de la connexion;
- la taille requise du tampon de connexion.

La taille du tampon de connexion doit inclure tout en-tête de transport.

Network Connection Parameters



Légende

Anglais	Français
Network Connection Parameters	Paramètres de connexion réseau
Redundant Owner	Propriétaire redondant
Size in Bytes	Taille en octets
Connection Fixed (0) or Variable (1)	Connexion fixe (0) ou variable (1)
Priority:	Priorité
Low Priority	Faible priorité
High Priority	Haute priorité
Scheduled Priority	Priorité programmée
Urgent	Urgent

Anglais	Français
Reserved	Réservé
Connection Type:	Type de connexion
Null	Vide
Multicast	Multidiffusion
Point to Point	Point à Point
Reserved	Réservé

Figure 4 – Paramètres de connexion réseau

4.1.6.1.2 Type de connexion

Le type de connexion doit être NULL, MULTIPOINT ou POINT2POINT. Le type NULL doit indiquer qu'aucune connexion réseau n'est indispensable, alors que MULTIPOINT et POINT2POINT renvoient aux types de connexion réseau tels que définis dans la CEI 61158-5-2. Le type de connexion NULL peut être utilisé pour reconfigurer une connexion.

4.1.6.1.3 Priorité

La priorité doit être LOW, HIGH, SCHEDULED ou URGENT, URGENT correspondant à la priorité la plus élevée (voir 4.1.6.3 pour le détail des spécifications et de l'utilisation).

Pour CP 2/1, les données SCHEDULED sont classées par ordre de priorité sur toute la liaison; cependant, les deux priorités non programmées (LOW et HIGH) sont arbitrées de façon indépendante en chaque nœud. Un nœud peut émettre un paquet de priorité LOW alors qu'un autre nœud a des données de priorité HIGH à envoyer.

4.1.6.1.4 Connection fixed/variable (Connexion fixe/variable)

La connexion peut être établie pour utiliser des connexions de taille fixe ou variable. Avec une connexion à taille fixe, chaque émission sur la connexion doit utiliser le tampon de même taille. Autrement, il doit se produire une condition soit de dépassement de capacité de tampon, soit de sous-utilisation de capacité de tampon. Avec une connexion de taille variable, chaque émission sur la connexion peut envoyer une quantité variable de données jusqu'à la taille maximale, qui doit être spécifiée lorsque la connexion est ouverte. Les sous-utilisations de capacité de tampon ne doivent pas être relatées, mais un dépassement de capacité de tampon peut toujours se produire si trop de données sont envoyées.

4.1.6.1.5 Taille de connexion

La taille de la connexion réseau doit être la taille du tampon requise pour la connexion, qui inclut les données et tout en-tête de transport. Pour une connexion de taille variable, la taille doit être la taille maximale du tampon pour un transfert quelconque. La taille réelle du transfert pour une connexion variable doit être inférieure ou égale à la taille spécifiée pour la connexion réseau. La taille de tampon maximale doit être fonction des liaisons que la connexion traverse.

4.1.6.1.6 Redundant owner (Propriétaire redondant)

Le bit "redundant owner" dans le sens O⇒T doit être mis (= 1) pour indiquer que plusieurs propriétaires peuvent être autorisés à établir simultanément une connexion. Le bit doit être effacé (= 0) pour indiquer une connexion en propriétaire exclusif, en saisie seule ou en écoute seule.

4.1.6.1.7 Reserved parameters (Paramètres réservés)

Les bits réservés doivent être effacés (= 0).

4.1.6.2 Intervalle réel et demandé entre paquets (CM_RPI et CM_API)

L'émetteur de la connexion indique le temps nécessaire entre deux productions pour chaque direction de la connexion. La cible indique le temps réel entre les productions. Quand une connexion traverse CP 2/1, le temps réel peut encore être réglé comme le montre 4.1.6.3.

L'intervalle demandé entre paquets (CM_RPI) doit être le temps demandé entre les paquets en microsecondes. Le format du CM_RPI doit être un nombre entier de 32 bits en microsecondes.

L'intervalle réel entre paquets (CM_API) doit être le temps maximal entre les paquets en microsecondes. L'API équivaut à la valeur Transmission Trigger Timer spécifiée dans la CEI 61158-5-2, 6.2.3.2.1.7. Le format du CM_API doit être un nombre entier de 32 bits en microsecondes.

L'effet des valeurs API et RPI sur le comportement de la connexion dépend de la classe de transport et des définitions du déclenchement données en 4.1.8.9.1.2. La valeur CM_RPI doit être utilisée pour attribuer une utilisation de la capacité à chaque nœud producteur. L'attribution de l'utilisation de la capacité peut devoir être réglée quand le CM_API est retourné puisque le CM_RPI et le CM_API sont susceptibles d'être différents.

Le CM_API doit être égal au CM_RPI dans les limites de résolution de temporisation de l'appareil, sauf pour le CP 2/1 comme l'indique 4.1.6.3. Si le CM_API n'est pas égal au CM_RPI, il doit être plus rapide que le CM_RPI sauf si le CM_API qui en résulte serait 0, dans ce cas le CM_RPI ne peut pas être pris en charge et une erreur est retournée, soit

- la ou les valeurs CM_RPI Not Acceptable (inacceptables), statut général 0x01, statut étendu 0x0112 (recommandé);
- CM_RPI non pris en charge, statut général 0x01, statut étendu 0x0111.

La valeur de temporisation Inactivity/Watchdog à chacun des nœuds intermédiaires et des nœuds cible doit être réglée à la valeur du multiplicateur de temporisation de connexion multiplié par le CM_RPI. Si la temporisation qui en résulte n'est pas réalisable dans les limites de la résolution du temporisateur de l'appareil, la temporisation doit être arrondie à la valeur suivante dans la résolution du temporisateur de l'appareil.

4.1.6.3 Fonction de priorités

Les priorités, dans l'ordre décroissant d'importance, sont les suivantes: urgent, programmée, élevée et faible.

Bien qu'il n'y ait pas d'exigences particulières concernant la mise en œuvre des appareils finaux pour obtenir une distinction interne du trafic selon différentes priorités, il est recommandé que les appareils traitent de préférence les connexions dont la priorité est élevée plutôt que les connexions dont la priorité est plus faible. Il est au minimum recommandé que les appareils donnent une priorité plus élevée au traitement des messages implicites qu'à celui des messages explicites.

La mise en correspondance des priorités CIP avec les priorités de liaison est particulière à chaque CP.

Priorité Urgent

La priorité Urgent doit uniquement être utilisée quand elle est explicitement définie dans les normes coordonnées de Type 2.

EXEMPLE La priorité Urgent est utilisée pour les connexions E/S établies dans l'objet Motion Device Axis spécifié dans la CEI 61800-7-202.

Sur CP 2/1:

- la largeur de bande doit être réservée en utilisant le segment Network Schedule;
- l'intervalle de données doit être restreint au CM_API, ce qui signifie que si des données arrivent au nœud intermédiaire plus vite que celles de l'API, le nœud doit filtrer les paquets à destination du CM_API sur CP 2/1;
- un outil chargé des opérations de programmation doit favoriser la priorité Urgent par rapport aux connexions à priorité programmée.

Sur les autres liaisons de Type 2:

- les nœuds doivent associer la priorité Urgent à la priorité de la liaison correspondante spécifique au réseau.

Priorité programmée

Sur CP 2/1:

- la largeur de bande doit être réservée en utilisant le segment Network Schedule;
- l'intervalle de données doit être restreint au CM_API, ce qui signifie que si des données arrivent au nœud intermédiaire plus vite que celles de CM_API, le nœud doit filtrer les paquets à destination du CM_API sur CP 2/1;
- un outil chargé des opérations de programmation doit favoriser la priorité Urgent par rapport aux connexions à priorité programmée.

Sur les autres liaisons de Type 2:

- les nœuds doivent associer la priorité programmée à la priorité de la liaison correspondante spécifique au réseau.

Priorité élevée

Les nœuds doivent associer la priorité élevée à la priorité correspondante de liaison spécifique au réseau.

Priorité faible

Les nœuds doivent associer la priorité faible à la priorité correspondante de liaison spécifique au réseau.

4.1.6.4 Connection Time-out Multiplier (Multiplicateur de temporisation de connexion)

Le Connection Time-out Multiplier spécifie le multiplicateur appliqué au CM_RPI pour obtenir la valeur de temporisation de connexion. L'appareil doit cesser d'émettre sur une connexion chaque fois que la connexion expire, même si la fermeture en attente a été envoyée. Le multiplicateur doit être tel que représenté dans le Tableau 37.

Tableau 37 – Multiplicateur de temporisation

Valeur	Multiplicateur	Notes
0	X 4	Valeur par défaut
1	X 8	
2	X 16	
3	X 32	
4	X 64	
5	X 128	

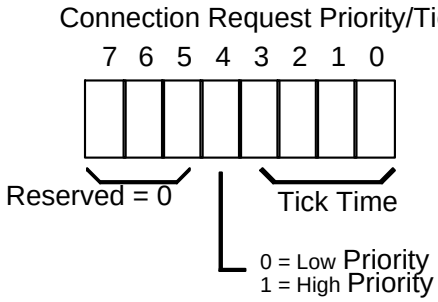
Valeur	Multiplicateur	Notes
6	X 256	
7	X 512	
8 à 255	réservé	

4.1.6.5 Connection request priority and tick time (Priorité de demande et temps de top d'horloge d'une connexion)

Deux valeurs sont condensées en ce champ. La priorité de demande de connexion (connection request priority) détermine la priorité des messages de l'UCMM utilisés pour établir la connexion et n'a pas de corrélation avec la priorité de la connexion réalisée. Bien que la faible priorité (Low priority) soit utilisée pour la plupart des établissements de connexions, la haute priorité (High priority) peut être utilisée pour les circonstances spéciales qui exigent qu'une connexion soit réalisée rapidement. L'utilisation de la faible priorité a été choisie pour éviter l'engorgement avec la messagerie à temps critique. Les messages de l'UCMM ne peuvent pas utiliser la priorité programmée.

Le temps de top d'horloge (tick time) doit être utilisé conjointement à la valeur des tops d'horloge de temporisation de demande de connexion. Cette valeur détermine le temps entre les "ticks" (tops d'horloge) dans le dernier champ. Ainsi, si le temps de top d'horloge est 1 ms, une valeur de temporisation de demande de connexion de 5 se traduira en 5 ms. Si le temps de top d'horloge était 2 ms, une valeur de temporisation de demande de connexion de 5 se traduirait en 10 ms.

Le format du Connection Request Priority/Tick Time doit être tel que montré à la Figure 5.



Légende

Anglais	Français
Connection Request Priority/Tick Time	Priorité de demande de connexion/Temps de top d'horloge
Reserved = 0	Réservé
Tick Time	Temps de top d'horloge
0 = Low Priority	Faible priorité
1 = High Priority	Haute priorité

Figure 5 – Temps de top d'horloge

Le Tableau 38 doit déterminer le temps entre tops.

Tableau 38 – Unités de top d'horloge

Temps de top		
Temps de top (binaire)	**Temps par top**	**Temps max.**
0000	1 ms	255 ms
0001	2 ms	510 ms
0010	4 ms	1 020 ms
0011	8 ms	2 040 ms
0100	16 ms	4 080 ms
0101	32 ms	8 160 ms
0110	64 ms	16 320 ms
0111	128 ms	32 640 ms
1000	256 ms	65 280 ms
1001	512 ms	130 560 ms
1010	1 024 ms	261 120 ms
1011	2 048 ms	522 240 ms
1100	4 096 ms	1 044 480 ms
1101	8 192 ms	2 088 960 ms
1110	16 384 ms	4 177 920 ms
1111	32 768 ms	8 355 840 ms

4.1.6.6 Connection request time-out ticks (Tops de temporisation de demande de connexion)

Les tops de temporisation de demande de connexion doivent être utilisés pour spécifier la quantité de temps que l'application émettrice doit attendre pour que la connexion soit établie. Chaque saut de la connexion a une valeur de temporisation spécifique à la liaison qui est mise en œuvre par l'UCMM et, après que les répétitions automatiques de tentative ont été épuisées sans acquittement, l'UCMM doit générer une erreur de temporisation. Par conséquent, si un appareil dans le chemin doit être soit absent, soit trop occupé pour acquitter une demande de l'UCMM, une temporisation d'UCMM doit se produire aussi rapidement que possible. Le seul cas de rencontre de la période de temporisation de demande de connexion doit être si le nœud cible acquitte la demande de connexion, mais échoue à répondre ou si la demande de connexion doit être délivrée, mais qu'une défaillance dans le réseau empêche que la réponse soit retournée. La temporisation de demande de connexion ne doit pas être la valeur de temporisation de chemin dérivée du CM_RPI et utilisée pour fermer une connexion après qu'elle doit être établie. La temporisation de demande de connexion doit être spécifiée par une combinaison des tops de temporisation de demande de connexion multipliés par le temps par top d'horloge qui doit être déterminé par la valeur du temps de top.

Sachant qu'une temporisation de demande de connexion particulière peut être exprimée en plusieurs formats différents, la règle doit être de **toujours la condenser de sorte à réduire la valeur du temps d'horloge au maximum.** Cela autorise la meilleure granularité pour diminuer la temporisation à mesure que le message est transmis par sauts. Cela signifie aussi que la valeur du top d'horloge peut varier à mesure que le message est transmis par sauts multiples. A titre d'exemple, le codage de 300 ms peut être:

1. 0001 10010110 -> Temps par top = 2 ms, Tops = 150 (correct)

2. 0010 01001011 -> Temps par top = 4 ms, Tops = 75 (erroné)

La valeur de temporisation de demande de connexion doit être fonction du nombre de segments de port dans le chemin de connexion, du temps de traitement au nœud cible et de l'engorgement des réseaux intermédiaires. La temporisation de demande de connexion peut être une valeur relativement élevée, car la pleine temporisation de demande de connexion ne doit se produire qu'en quelques rares cas. La seule fois que se produirait la pleine temporisation de demande de connexion serait si:

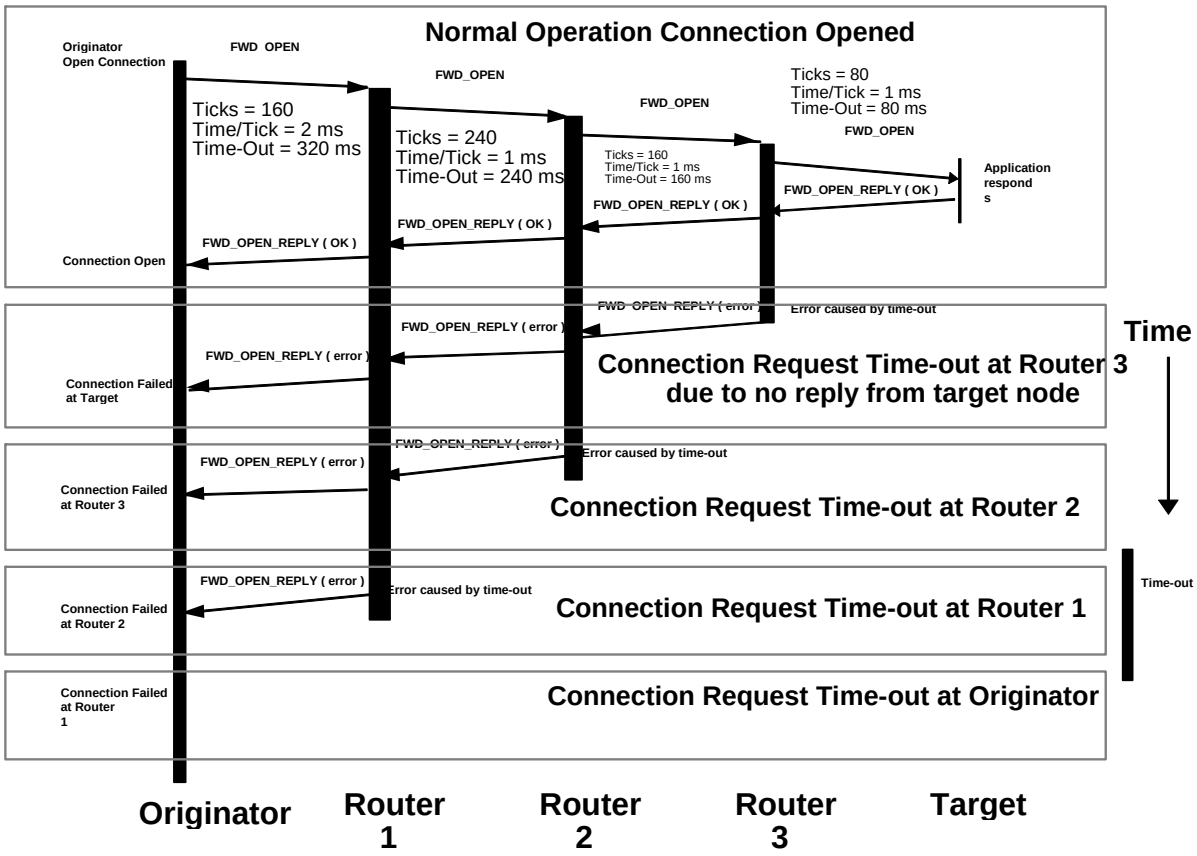
- la demande de connexion avait été délivrée avec succès au nœud cible et le nœud avait échoué à générer une réponse. Cela serait habituellement provoqué par une erreur de firmware dans l'appareil;
- la demande de connexion avait été délivrée avec succès au nœud cible et la réponse avait été rejetée en raison d'un engorgement au nœud intermédiaire;
- la demande de connexion avait été délivrée avec succès au nœud cible et le nœud cible ou l'un des nœuds intermédiaires avait été retiré ou mis hors tension avant que la réponse ne fût retournée.

L'UCMM doit être utilisé pour transmettre la demande de connexion vers le prochain nœud du chemin. L'UCMM envoie la demande et attend un acquittement ou une réponse pendant un temps spécifique à la liaison; si aucun acquittement ou aucune réponse ne doit être reçu(e) avant la temporisation spécifique à la liaison, la demande doit être répétée. Après un nombre (trois en général) de répétitions de tentative spécifique à la liaison, l'UCMM doit signaler une temporisation sur la demande de connexion. Le Connection Manager qui a généré la demande doit retourner une réponse d'erreur à la demande de connexion. Par conséquent, la demande de connexion doit s'arrêter aussitôt que possible pour les défauts de système les plus courants

- appareils absents ou mis hors tension dans le chemin;
- engorgement à l'appareil dans le chemin.

Pour aider au débogage des systèmes, chaque saut dans le chemin doit décrémenter la temporisation de 80 ms avant de l'expédier dans le Forward_Open vers le prochain appareil du chemin. Chaque appareil intermédiaire (routeur) doit arrondir la valeur de temporisation de connexion qui doit être décrémentée de sorte qu'un minimum de 1 top doit être soustrait. Cela doit permettre au dernier appareil avant la défaillance de temporiser le premier et retourner les informations relatives au point en lequel la connexion a été faite avant la défaillance. Ces informations peuvent être utilisées pour aider au débogage de problèmes dans un système. Une fonction système typique est montrée à la Figure 6.

Connection Request Time-out in Multi-hop Systems



Légende

Anglais	Français
Connection Request Time-out in Multi-hop Systems	Temporisation de demande de connexion dans les systèmes à sauts multiples
Normal Operation Connection Opened	Fonctionnement normal connexion ouverte
Originator	Emetteur
Open Connection	Ouvrir connexion
Ticks	Tops
Time/Tick	Temps/top
Time-Out	Temporisation
Application responds	L'application répond
Connection Open	Connexion ouverte
Connection Failed at Target	Echec de connexion au niveau de la cible
Error caused by time-out	Erreur pour cause de temporisation
Time	Temps
Connection Request Time-out at Router 3 due to no reply from target node	Temporisation de demande de connexion au niveau du routeur 3 en raison de l'absence de réponse de la part du nœud cible
Connection Failed at Router 3	Echec de connexion au niveau du routeur 3
Connection Failed at Router 2	Echec de connexion au niveau du routeur 2
Connection Failed at Router 1	Echec de connexion au niveau du routeur 1
Connection Request Time-out at Router 2	Temporisation de demande de connexion au niveau du routeur 2
Connection Request Time-out at Router 1	Temporisation de demande de connexion au niveau du routeur 1

Anglais	Français
Connection Request Time-out at Originator	Temporisation de demande de connexion au niveau de l'émetteur
Originator Router	Routeur émetteur
Router	Routeur
Target	Cible

Figure 6 – Temporisation d'établissement de connexion

A la Figure 6, une connexion est demandée sur un chemin contenant 4 sauts. La Connection request Time-out est spécifiée comme étant de 320 ms par l'émetteur de la connexion. Dans le premier cas, la connexion est parachevée et les valeurs de temporisation calculées sont montrées en chaque saut dans le chemin. Le deuxième cas est celui où le dernier appareil reçoit un acquittement, mais ne reçoit pas de réponse et temporise. Il retourne une erreur de temporisation à l'émetteur. Comme la valeur de temporisation décroît pour chaque saut, l'erreur de temporisation doit être retournée à l'émetteur avant que l'émetteur ne temporise.

Les autres cas montrent des temporisations en les autres appareils et dans chaque cas, l'erreur de temporisation doit être retournée à l'émetteur avant que l'émetteur lui-même ne temporise. Le dernier cas est celui où aucune réponse n'est reçue par l'émetteur et le processus d'ouverture de connexion a expiré. La raison de faire décroître la temporisation en chaque nœud est que peuvent être ainsi envoyées des informations d'erreur supplémentaires indiquant à quel endroit dans le chemin la temporisation a eu lieu. Se rappeler que si l'un quelconque des appareils ne reçoit pas un acquittement, la demande de connexion doit immédiatement être temporisée et une réponse d'erreur doit être générée.

4.1.6.7 Connection serial number (Numéro de série de connexion)

Le numéro de série de connexion (connection serial number) doit être une valeur unique de 16 bits sélectionnée par le Connection Manager au niveau de l'émetteur de la connexion. L'émetteur doit s'assurer que la valeur de 16 bits est unique pour l'appareil. Aucune autre signification ne doit être placée sur le numéro par un quelconque autre nœud dans le chemin de connexion. Les numéros de série de connexion doivent être uniques, mais peuvent ne pas être séquentiels. Par exemple, une interface opérateur peut avoir un grand nombre de connexions ouvertes en même temps, chacune avec un numéro unique. Les mêmes valeurs pourraient être répétées en d'autres postes de l'interface opérateur. Une mise en œuvre possible serait d'avoir une liste de connexions qui pointe vers le descripteur pour chaque connexion et le numéro de série de connexion pourrait être l'indice dans le tableau.

4.1.6.8 ID de vendeur

Le numéro de vendeur doit être un nombre unique attribué aux divers vendeurs de produits. Chaque vendeur a un numéro unique attribué.

NOTE Les ID de vendeur sont attribués par ODVA, Inc.

4.1.6.9 Originator serial number (Numéro de série d'émetteur)

Le numéro de série d'émetteur (originator serial number) doit être une valeur unique de 32 bits qui est attribuée à un appareil au moment de la fabrication. Cette valeur doit être garantie comme étant unique pour tous les appareils fabriqués par le même vendeur. Aucune signification ne doit être attachée au numéro. La combinaison de l'ID de vendeur et de l'Originator Serial Number doit être unique dans le système tout entier.

4.1.6.10 Connection number (Numéro de connexion)

Le numéro de connexion (connection number) doit être une valeur de 16 bits qui est attribuée par le Connection Manager lorsqu'une connexion est ouverte. Cette valeur permet à d'autres nœuds d'obtenir des données de connexion délivrées par le Connection Manager. Le numéro ne doit pas être confondu avec le Connection Serial Number (numéro de série de connexion).

4.1.6.11 Connection path size (Taille de chemin de connexion)

La taille de chemin de connexion (connection path size) doit être la longueur du chemin de connexion en mots de 16 bits. La longueur du chemin de connexion varie au cours du processus de connexion, car chaque nœud sur le chemin de connexion enlève le segment de port courant et transmet seulement les segments de chemin restants vers le prochain nœud.

4.1.6.12 Chemin de connexion / chemin de connexion restant

Le paramètre de chemin de connexion doit contenir un ou plusieurs chemins codés d'après ce que demande la combinaison du service et de la valeur des autres paramètres à l'intérieur des données de service. L'itinéraire et les informations liées (comme le port, le réseau, les segments de clé électronique), s'ils existent, doivent être prioritaires par rapport au(x) chemin(s) de l'application comme on le voit ci-dessous.

Le chemin de connexion doit prendre la forme suivante:

[[segment1 Electronic Key] [segment11 Network]...[segment1N Network] segment1 Port]	} } Port Segment Group pour le 1er nœud intermédiaire }
[[segment2 Electronic Key] [segment21 Network]...[segment2N Network] Segment2 de port]	} } Port Segment Group pour le 2e nœud intermédiaire }
...	
[[segmentM Electronic Key] [segmentM1 Network]...[segmentMN Network] segmentM Port]	} } Port Segment Group pour le Me nœud intermédiaire }
[segmentP Electronic Key] [segmentP1 Network]...[segmentPN Network] Application Path1 [Application Path2] [Application Path3] [Simple Data segment]	} } } Application Segment Group pour la cible } }

Tous les segments entre crochets [...] sont facultatifs. L'objet Connection Manager du nœud intermédiaire et du nœud cible doit traiter tous les segments qui lui sont adressés.

Les segments de réseau sont admis à des endroits spécifiques uniquement s'ils sont admis par la définition du segment de réseau spécifique.

Si un segment de données de configuration fait partie du chemin de configuration, le segment de données de configuration doit suivre le ou les chemins d'application.

Les composants d'un chemin de connexion sont spécifiés de 4.1.9.1 à 4.1.9.7 et la hiérarchie d'un chemin de connexion est spécifiée en 4.1.9.8.

Suivant les champs O2T_connection_parameters et T2O_connection_parameters et la présence d'un segment de données, il faut spécifier un ou plusieurs chemins codés d'application. En général, les chemins d'application suivent l'ordre du chemin de configuration, du chemin de consommation et du chemin de production. On peut cependant utiliser un chemin unique codé quand la configuration, la consommation et/ou la production utilisent le

même chemin. Voir le Tableau 39 qui indique les combinaisons valides et la signification implicite des chemins d'application. Les chemins d'application sont liés au nœud cible.

Tableau 39 – Ordre du chemin d'application codé

Paramètres de connexion réseau		Segment de données présent	Nombre de chemins d'application codés		
Type de connexion O2T	Type de connexion T2O		1	2	3
NULL	NULL	Oui	Le chemin sert à la configuration	Le premier chemin sert à la configuration, le deuxième chemin est ignoré	Le premier chemin sert à la configuration, le deuxième et le troisième chemins sont ignorés
		Non	Le chemin sert à envoyer un ping à un appareil [a]	Invalid	Invalid
non NULL	NULL	Oui	Le chemin sert à la configuration et à la consommation	Le premier chemin sert à la configuration, le deuxième à la consommation	Invalid
		Non	Le chemin sert à la consommation	Invalid	Invalid
NULL	non NULL	Oui	Le chemin sert à la consommation et à la production	Le premier chemin sert à la configuration, le deuxième chemin à la production	Invalid
		Non	Le chemin sert à la production	Invalid	Invalid
non NULL	non NULL	Oui	Le chemin sert à la configuration, à la consommation et à la production	Le premier chemin sert à la configuration, le deuxième à la consommation et à la production	Le premier chemin sert à la configuration, le deuxième à la consommation et le troisième à la production
		Non	Le chemin sert à la consommation et à la production	Le premier chemin sert à la consommation, le deuxième à la production	Le premier chemin est ignoré, le deuxième chemin sert à la consommation et le troisième chemin à la production

[a] Si le chemin "20 01 24 01" est utilisé pour envoyer un ping à un appareil, voir 4.1.5.3.2.3 (null Forward_Open, non concordant), sinon invalide.

Les règles de compression du chemin codé figurent en 4.1.9.9.

4.1.6.13 Network connection ID (ID de connexion réseau)

L'ID de connexion réseau (Network Connection ID) doit être spécifique à une liaison et ne doit pas être relié au numéro de série de connexion, qui est spécifique à une connexion et le même sur toutes les liaisons. Les champs du Network Connection ID doivent être utilisés pour établir le mécanisme de filtrage pour la liaison spécifiée.

L'ID de connexion réseau est soit un ID de connexion produite soit un ID de connexion consommée.

Un ID de connexion multidiffusion ne doit pas être réutilisé jusqu'à ce que toutes les connexions associées à l'ID de connexion aient été fermées ou que le délai se soit écoulé.

L'ID de connexion réseau est spécifié plus en détail dans les DMPM correspondant aux différentes couches de liaison qui peuvent être associées à la couche d'application de Type 2 (voir Article 10, Article 11 et Article 12).

4.1.6.14 Transport class and trigger (Classe de transport et déclencheur)

La classe de transport et le déclencheur (transport class and trigger) spécifient le type de transport et le déclencheur de production requis pour la connexion. L'octet qui code la classe de transport et le déclencheur doit avoir la forme suivante montrée dans le Tableau 40.

Tableau 40 – Format de classe de transport, déclencheur et Is_Server

TransportClass (4 bits)	TriggerMode (3 bits)	Is_Server (1 bit)
NULL = 0 DUPLICATE_DETECT = 1 ACKNOWLEDGED = 2 VERIFIED = 3 NONBLOCKING = 4 FRAGMENTING = 5 MULTIPOINT_FRAG = 6	CYCLIC = 0 CHANGE_OF_STATE = 1 APPLICATION = 2	IS_NOT_SERVER = 0 IS_SERVER = 1

Les 4 bits de poids faible, `class` (classe), doivent déterminer quel type de transport est spécifié et doivent être situés dans la plage de 0 à 6. Les valeurs de `class` 7 à 15 doivent être réservées. Les bits 4 à 6, appelés `trigger` (déclencheur), doivent déterminer quel événement provoque la production de données sur la connexion et doivent être situés dans la plage de 0 à 2. Les valeurs de `trigger` 3 à 7 doivent être réservées. Pour les transports qui ont besoin de spécifier la cible comme un serveur, le champ `is_server` doit être 1; autrement, il doit être 0. Pour les classes de transport 0 et 1, le champ `is_server` doit être ignoré. Le bit 7 doit être le champ `is_server`.

Voir la CEI 61158-5-2 pour connaître les détails de comportement des différentes combinaisons de `is_server`, `class` et `trigger`, CM_RPI et CM_API.

4.1.7 En-têtes de MR

4.1.7.1 Paquets de demande de MR

Tous les paquets de demande livrés au Message Router, soit en provenance d'une connexion, soit en provenance de l'UCMM, doivent avoir un en-tête de la forme montrée dans le Tableau 41.

Tableau 41 – Format de MR_Request_Header

Paramètre	Format
Service	USINT
path_size	USINT
path	Padded EPATH

Le premier octet, service, doit être livré directement à l'objet d'application et doit se situer dans la plage de 1 à 127. Le paramètre `path_size` doit déterminer le nombre de mots de 16 bits contenus dans le champ `path`.

Le champ `path` doit contenir les informations d'adressage associées au paquet (chemin d'application) et d'autres informations, et doit être bourré.

Le champ `path` doit avoir la forme suivante:

[segment Electronic Key]

Application Path

Le segment Electronic Key est conditionnel. Si la demande est envoyée à travers une connexion, le segment Electronic Key n'est pas autorisé; autrement, il est facultatif. Si le segment Electronic Key est présent, la cible doit évaluer la clé. Voir 4.1.9.4.2 pour consulter la définition du segment Electronic Key et 4.1.9.8 pour consulter celle du format Application Path.

EXEMPLE

34 04 42 42 0C 00 01 00 81 01	Le segment Electronic Key correspondant à l'appareil cible doit être compatible avec un appareil possédant les caractéristiques suivantes: Vendor (Vendeur) 0x4242, Device Type (Type d'appareil) 0x000C, Product Code (Code produit) 0x0001, Major Revision (Révision majeure) 0x01, Minor Revision (Révision mineure) 0x01.
20 04 24 01 30 03	Chemin d'application qui spécifie l'Instance 1 de l'objet Assembly, Attribut 3.

4.1.7.2 Paquets de réponse MR

Les paquets de réponse issus du Message Router doivent avoir un en-tête de la forme montrée dans le Tableau 42.

Tableau 42 – Format de MR_Response_Header

Paramètre	Format
Service	USINT
Réservé	USINT
general_status	USINT
Extended_status_size	USINT
Extended_status[]	WORD

Le champ `service` doit être la valeur du champ `MR_Request_Header.service` avec son bit de poids le plus fort mis (plus 0x80). Si `MR_Request_Header.service` est égal à 0x05, le champ `MR_Response_Header.service` doit être égal à 0x85. Le champ `general_status` doit être rempli à l'aide du paramètre `Status Code` de la réponse de service. De même, les paramètres `extended_status`, `extended_status_size`, et `response` doivent respectivement être remplis à l'aide du paramètre `Extended Status` et des autres paramètres de réponse issus de la réponse de service. Si le paramètre `extended_status_size` est égal à 0, il n'y a pas de paramètre `extended_status`. Le `response_size` ne doit pas être inclus de façon explicite dans le `MR_Response_Header`; il doit être sous-entendu par le nombre d'octets de `MR_Response_Header`.

4.1.8 OM_Service_PDU

4.1.8.1 Syntaxe générale

4.1.8.1.1 Get_Attribute_All

Le corps de Get_Attribute_All_RequestPDU est vide.

Le corps de Get_Attribute_All_ResponsePDU doit être tel que spécifié dans le Tableau 43.

Tableau 43 – Structure du corps de Get_Attribute_All_ResponsePDU

Nom	Type de données
Attribute Data	Spécifique à un attribut d'objet/de classe

4.1.8.1.2 Set_Attribute_All

Le corps de Set_Attribute_All_RequestPDU doit être tel que spécifié dans le Tableau 44.

Tableau 44 – Structure du corps de Set_Attribute_All_RequestPDU

Nom	Type de données
Attribute Data	Spécifique à un attribut d'objet/de classe

Le corps de Get_Attribute_All_ResponsePDU est vide.

4.1.8.1.3 Get_Attribute_List

Le corps de Get_Attribute_List_RequestPDU doit être tel que spécifié dans le Tableau 45.

Tableau 45 – Structure du corps de Get_Attribute_List_RequestPDU

Nom	Type de données
Attribute Count	UINT
Attribute List	Array of UINT

Le corps de Get_Attribute_List_ResponsePDU doit être tel que spécifié dans le Tableau 46.

Tableau 46 – Structure du corps de Get_Attribute_List_ResponsePDU

Nom	Type de données	Sémantique des valeurs
Attribute Count	UINT	
Attribute Data	Array of STRUCT	
Attribute Identifier	UINT	
Attribute Status	UINT	Les valeurs 0 à 255 sont réservées à des codes de statut de services communs mineurs. Les valeurs 256 à 65 535 sont disponibles pour les erreurs spécifiques à un attribut d'objet/de classe.
Attribute Value	Spécifique à un attribut d'objet/de classe	

4.1.8.1.4 Set_Attribute_List

Le corps de Set_Attribute_List_RequestPDU doit être tel que spécifié dans le Tableau 47.

Tableau 47 – Structure du corps de Set_Attribute_List_RequestPDU

Nom	Type de données
Attribute Count	UINT
Attribute Data	Array of STRUCT
Attribute Identifier	UINT

Nom	Type de données
Attribute Value	Spécifique à un attribut d'objet/de classe

Le corps de Set_Attribute_List_ResponsePDU doit être tel que spécifié dans le Tableau 48.

Tableau 48 – Structure du corps de Set_Attribute_List_ResponsePDU

Nom	Type de données	Sémantique des valeurs
Attribute Count	UINT	
Attribute Status List	Array of STRUCT	
Attribute Identifier	UINT	
Attribute Status	UINT	Les valeurs 0 à 255 sont réservées à des codes de statut de services communs mineurs. Les valeurs 256 à 65 535 sont disponibles pour les erreurs spécifiques à un attribut d'objet/de classe.

4.1.8.1.5 Reset

Le corps de Reset_RequestPDU doit être tel que spécifié dans le Tableau 49, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la demande Reset est spécifié. Sinon, il doit être vide.

Tableau 49 – Structure du corps de Reset_RequestPDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Reset_ResponsePDU doit être tel que spécifié dans le Tableau 50, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Reset est spécifié. Sinon, il doit être vide.

Tableau 50 – Structure du corps de Reset_ResponsePDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.6 Start

Le corps de Start_RequestPDU doit être tel que spécifié dans le Tableau 51, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la demande Start est spécifié. Sinon, il doit être vide.

Tableau 51 – Structure du corps de Start_RequestPDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Start_ResponsePDU doit être tel que spécifié dans le Tableau 52, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Start est spécifié. Sinon, il doit être vide.

Tableau 52 – Structure du corps de Start_ResponsePDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.7 Stop

Le corps de Stop_RequestPDU doit être tel que spécifié dans le Tableau 53, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la demande Stop est spécifié. Sinon, il doit être vide.

Tableau 53 – Structure du corps de Stop_RequestPDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Stop_ResponsePDU doit être tel que spécifié dans le Tableau 54, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Stop est spécifié. Sinon, il doit être vide.

Tableau 54 – Structure du corps de Stop_ResponsePDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.8 Create

Le corps de Create_RequestPDU doit être tel que spécifié dans le Tableau 55, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la demande Create est spécifié. Sinon, il doit être vide.

Tableau 55 – Structure du corps de Create_RequestPDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Create_ResponsePDU doit être tel que spécifié dans le Tableau 56, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Create est spécifié. Sinon, il doit être vide.

Tableau 56 – Structure du corps de Create_ResponsePDU

Nom	Type de données
Instance ID	UINT
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.9 Delete

Le corps de Delete_RequestPDU doit être tel que spécifié dans le Tableau 57, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la demande Delete est spécifié. Sinon, il doit être vide.

Tableau 57 – Structure du corps de Delete_RequestPDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Delete_ResponsePDU doit être tel que spécifié dans le Tableau 58, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Delete est spécifié. Sinon, il doit être vide.

Tableau 58 – Structure du corps de Delete_ResponsePDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.10 Get_Attribute_Single

Le corps Get_Attribute_Single_RequestPDU est vide, sauf dans les cas ci-dessous.

Certains profils CPF 2 prennent en charge des formats de message explicites spécifiques à la liaison (c'est-à-dire qu'ils ne prennent pas en charge le format de la demande de Message Router), ce qui ne permet de spécifier l'Attribute ID dans le paramètre Path. Pour ces formats, l'Attribute ID est limité à USINT et il est transmis sous forme de corps Get_Attribute_Single_ResponsePDU.

Le corps de Get_Attribute_ Single_ResponsePDU doit être tel que spécifié dans le Tableau 59.

Tableau 59 – Structure du corps de Get_Attribute_Single_ResponsePDU

Nom	Type de données
Attribute Data	Spécifique à un attribut d'objet/de classe

4.1.8.1.11 Set_Attribute single

Le corps de Set_Attribute_Single_RequestPDU doit être tel que spécifié dans le Tableau 60.

Tableau 60 – Structure du corps de Set_Attribute_Single_RequestPDU

Nom	Type de données
Attribute Data	Spécifique à un attribut d'objet/de classe

Certains profils CPF 2 prennent en charge des formats de message explicites spécifiques à la liaison (c'est-à-dire qu'ils ne prennent pas en charge le format de la demande de Message Router), ce qui ne permet de spécifier l'Attribute ID dans le paramètre Path. Pour ces formats, l'Attribute ID est limité à USINT et il est transmis sous la forme du premier paramètre (octet) du corps Set_Attribute_Single_RequestPDU.

Le corps de Set_Attribute_Single_ResponsePDU doit être tel que spécifié dans le Tableau 61, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Set_Attribute_Single est spécifié. Sinon, il doit être vide.

Tableau 61 – Structure du corps de Set_Attribute_Single_ResponsePDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.12 Find_Next_Object_Instance

Le corps de Find_Next_Object_Instance_RequestPDU doit être tel que spécifié dans le Tableau 62.

Tableau 62 – Structure du corps de Find_Next_Object_Instance_RequestPDU

Nom	Type de données
Maximum Returned Values	USINT

Le corps de Find_Next_Object_Instance_ResponsePDU doit être tel que spécifié dans le Tableau 63.

Tableau 63 – Structure du corps de Find_Next_Object_Instance_ResponsePDU

Nom	Type de données
Number Of List Members	USINT
Instance ID List	Array of UINT

4.1.8.1.13 NOP

Le corps de NOP_RequestPDU est vide.

Le corps de NOP_ResponsePDU est vide.

4.1.8.1.14 Apply_Attributes

Le corps de Apply_Attributes_RequestPDU doit être tel que spécifié dans le Tableau 64, si le paramètre de service **facultatif** “Object Specific Data” (données spécifiques à un objet) de la demande Apply_Attributes est spécifié. Sinon, il doit être vide.

Tableau 64 – Structure du corps de Apply_Attributes_RequestPDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Apply_Attributes_ResponsePDU doit être tel que spécifié dans le Tableau 65, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Apply_Attributes est spécifié. Sinon, il doit être vide.

Tableau 65 – Structure du corps de Apply_Attributes_ResponsePDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.15 Save

Le corps de Save_RequestPDU doit être tel que spécifié dans le Tableau 66, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la demande Save est spécifié. Sinon, il doit être vide.

Tableau 66 – Structure du corps de Save_RequestPDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Save_ResponsePDU doit être tel que spécifié dans le Tableau 67, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Save est spécifié. Sinon, il doit être vide.

Tableau 67 – Structure du corps de Save_ResponsePDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.16 Restore

Le corps de Restore_RequestPDU doit être tel que spécifié dans le Tableau 68, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la demande Restore est spécifié. Sinon, il doit être vide.

Tableau 68 – Structure du corps de Restore_RequestPDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Restore_ResponsePDU doit être tel que spécifié dans le Tableau 69, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Restore est spécifié. Sinon, il doit être vide.

Tableau 69 – Structure du corps de Restore_ResponsePDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.17 Get_Member

Le corps de Get_Member_RequestPDU est vide.

Le corps de Get_Member_RequestPDU doit être tel que spécifié dans le Tableau 70.

Tableau 70 – Structure du corps de Get_Member_ResponsePDU

Nom	Type de données
Member ID Member data	Voir 4.1.8.1.21 pour plus de détails

4.1.8.1.18 Set_Member

Le corps de Set_Member_RequestPDU doit être tel que spécifié dans le Tableau 71.

Tableau 71 – Structure du corps de Set_Member_RequestPDU

Nom	Type de données
Member data	Voir 4.1.8.1.21 pour plus de détails

Le corps de Set_Member_ResponsePDU doit être tel que spécifié dans le Tableau 72, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Set_Member est spécifié. Sinon, il doit être vide.

Tableau 72 – Structure du corps de Set_Member_ResponsePDU

Nom	Type de données
Member ID Member data	Voir 4.1.8.1.21 pour plus de détails

4.1.8.1.19 Insert_Member

Le corps de Insert_Member_ResponsePDU doit être tel que spécifié dans le Tableau 73, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la demande Insert_Member est spécifié. Sinon, il doit être vide.

Tableau 73 – Structure du corps de Insert_Member_RequestPDU

Nom	Type de données
Member data	Voir 4.1.8.1.21 pour plus de détails

Le corps de Insert_Member_ResponsePDU doit être tel que spécifié dans le Tableau 74. Si le paramètre de service **facultatif** "Member Data" de la réponse Insert_Member n'est pas spécifié, ce champ doit être vide.

Tableau 74 – Structure du corps de Insert_Member_ResponsePDU

Nom	Type de données
Member ID Member data	Voir 4.1.8.1.21 pour plus de détails

4.1.8.1.20 Remove_Member

Le corps de Remove_Member_RequestPDU est vide.

Le corps de Remove_Member_RequestPDU doit être tel que spécifié dans le Tableau 75.

Tableau 75 – Structure du corps de Remove_Member_ResponsePDU

Nom	Type de données
Member ID Member data	Voir 4.1.8.1.21 pour plus de détails

4.1.8.1.21 Eléments communs de syntaxe pour les services _Member

4.1.8.1.21.1 Généralités

Les attributs des classes d'objet peuvent être des matrices de types de données ou de structures de base. Pour manipuler les membres d'une matrice, les services suivants de membres ont recours aux éléments communs de syntaxe définis en 4.1.8.1.21:

- Get_Member;
- Set_Member;
- Insert_Member;
- Remove_Member.

Le premier membre d'une matrice est spécifié par une valeur Member ID de un (1).

4.1.8.1.21.2 Description de Member ID/EX

Deux formats de message sont définis pour les services de membre: format de base et format étendu. Le format étendu comprend jusqu'à 225 options de protocole. Le format du message est choisi d'après le bit le plus significatif du Member ID

- Quand un sous-réseau ne prend pas en charge l'adressage du niveau Attribute et Member, le paramètre Member ID/EX est envoyé sous forme de paramètre de 16 bits (WORD) dans les données de service.
- Si le sous-réseau prend en charge l'adressage du niveau Attribute et Member, le paramètre Member ID/EX est envoyé dans le Logical Segment (segment logique) et peut avoir 8, 16 ou 32 bits (SWORD, WORD, DWORD).

La Figure 7 définit les bits qui se trouvent dans le champ Member ID/EX quand il est envoyé sous forme de WORD.

Bits							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bits 0 à 7 de Member ID							
EX	Bits 8 à 14 de Member ID						

Figure 7 – Description de Member ID/EX (WORD)

Les bits de poids faible (de 0 à 14 pour un mot, WORD) du champ Member ID/EX identifient le membre à manipuler. Le bit de poids le plus élevé EX (15 pour un mot, WORD) détermine le format du message, soit le format de base (EX=0) soit le format étendu (EX=1).

4.1.8.1.21.3 Demande du membre – format de base

Le Tableau 76 donne la structure commune de base du corps _RequestPDU pour des services Member qui utilisent le format de base.

Tableau 76 – Structure commune du corps _Member_RequestPDU (format de base)

Nom	Type de données	Description du paramètre
ID d'attribut [a]	USINT	Identifie l'attribut du membre bénéficiaire du service.
Member ID/ EX [b]	WORD	Les 15 bits de poids faible identifient le Member ID du membre bénéficiaire du service. Si la valeur est zéro (0), le service utilisé détermine le comportement. Option de protocole étendu EX = 0 (bit 15)
Member Data (conditionnel)	Spécifique à un membre	Ce champ est nécessaire à certains services, voir la description individuelle des services.

a	Ce paramètre doit exister quand la cible de la demande se trouve dans un sous-réseau qui ne prend pas en charge l'adressage de message explicite au niveau de l'attribut. Si le sous-réseau prend en charge l'adressage au niveau de l'attribut, ce paramètre ne doit pas exister. Dans ce dernier cas, le type de données de l'Attribute ID doit être USINT, UINT, ou UDINT.
b	Ce paramètre doit exister quand la cible de la demande se trouve dans un sous-réseau qui ne prend pas en charge l'adressage de message explicite au niveau de l'attribut. Si le sous-réseau ne prend pas en charge l'adressage au niveau de l'attribut et du membre, ce paramètre ne doit pas exister. Dans ce dernier cas, le type de données de Member ID/EX peut être SWORD, WORD ou DWORD, le bit le plus significatif étant le paramètre Extended Protocol Option et les bits restants le Member ID.

Le Tableau 77 donne la structure commune de base du corps _ResponsePDU pour les services Member qui utilisent le format de base.

Tableau 77 – Structure commune du corps _Member_ResponsePDU (format de base)

Nom	Type de données	Description du paramètre
Member ID (conditionnel)	UINT	Le Member ID du membre bénéficiaire du service Ce champ est nécessaire à certains services, voir la description individuelle des services. Ce champ est nécessaire si le champ Member Data existe.
Member Data (conditionnel)	Spécifique à un membre	Ce champ est nécessaire à certains services, voir la description individuelle des services.

4.1.8.1.21.4 Demande de membre - format étendu (généralités)

Si le bit de protocole étendu [EX] du champ Member ID/EX a pour valeur un, l'octet qui suit le Member ID/EX définit le reste du protocole.

La plage des valeurs pour le Extended Protocol ID se divise en trois catégories (ouvert, spécifique au vendeur, réservé) comme l'indique 4.1.10.1.1.

Le Tableau 78 donne la structure commune de base du corps _RequestPDU pour des services Member qui utilisent le format étendu.

Tableau 78 – Structure commune du corps _Member_RequestPDU (format étendu)

Nom	Type de données	Description du paramètre
Attribute ID [a]	USINT	Identifie l'attribut du membre bénéficiaire du service.
Member ID/ EX [b]	WORD	Les 15 bits de poids faible identifient le Member ID du membre bénéficiaire du service. Si la valeur est zéro (0), le service utilisé détermine le comportement. Option de protocole étendu EX = 1 (bit 15)
Extended Protocol ID (Identificateur de protocole étendu)	USINT	Sélectionne le protocole étendu utilisé pour le reste du message. Voir Tableau 80.
Données supplémentaires	Spécifique au protocole	Les données supplémentaires de demande sont définies par le protocole étendu.

a	Ce paramètre doit exister quand la cible de la demande se trouve dans un sous-réseau qui ne prend pas en charge l'adressage de message explicite au niveau de l'attribut. Si le sous-réseau prend en charge l'adressage au niveau de l'attribut, ce paramètre ne doit pas exister. Dans ce dernier cas, le type de données de l'Attribute ID doit être USINT, UINT, ou UDINT.
b	Ce paramètre doit exister quand la cible de la demande se trouve dans un sous-réseau qui ne prend pas en charge l'adressage de message explicite au niveau de l'attribut. Si le sous-réseau ne prend pas en charge l'adressage au niveau de l'attribut et du membre, ce paramètre ne doit pas exister. Dans ce dernier cas, le type de données de Member ID/EX peut être SWORD, WORD ou DWORD, le bit le plus significatif étant le paramètre Extended Protocol Option et les bits restants le Member ID.

Le Tableau 79 donne la structure commune de base du corps _ResponsePDU pour les services Member qui utilisent le format étendu.

Tableau 79 – Structure commune du corps _Member_ResponsePDU (format étendu)

Nom	Type de données	Description du paramètre
Données de réponse	Spécifique au protocole	Les données de réponse sont définies par l'option du protocole étendu.

Le Tableau 80 donne les valeurs couramment définies pour l'Extended Protocol ID.

Tableau 80 – Extended Protocol ID (identificateur de protocole étendu)

Valeur (hex)	Description
00	Ouvert
01	Multiple Sequential Members (Membres séquentiels multiples)
02	Sélection de chaîne internationale
03 à 63	Open (Ouverts)
64 à C7	Vendor specific (Spécifique à un vendeur)
C8 à FF	Réservé
NOTE Les plages réservées peuvent être définies par ODVA, Inc.	

4.1.8.1.21.5 Demande d'un membre – format étendu (Membres séquentiels multiples)

Quand le Extended Protocol ID prend pour valeur Multiple Sequential Members [1], la structure du corps _RequestPDU est donnée au Tableau 81.

Tableau 81 – Structure du corps _Member_RequestPDU (Multiple Sequential Members, Membres séquentiels multiples)

Nom	Type de données	Description du paramètre
Attribute ID [a]	USINT	Identifie l'attribut du membre bénéficiaire du service.
Member ID/ EX [b]	WORD	Les 15 bits de poids faible identifient le Member ID du membre bénéficiaire du service. Si la valeur est zéro (0), le service utilisé détermine le comportement. Option de protocole étendu EX = 1 (bit 15)
Extended Protocol ID (Identificateur de protocole étendu)	USINT	Sélectionne l'option de protocole Multiple Sequential Members (valeur =1)
Nombre de membres	UINT	Nombre de membres bénéficiaires du service
Member Data (conditionnel)	Spécifique à un membre	Données du membre séquentiel multiple Ce champ est nécessaire à certains services, voir la description individuelle des services.

[a] Ce paramètre doit exister quand la cible de la demande se trouve dans un sous-réseau qui ne prend pas en charge l'adressage de message explicite au niveau de l'attribut. Si le sous-réseau prend en charge l'adressage au niveau de l'attribut, ce paramètre ne doit pas exister. Dans ce dernier cas, le type de données de l'Attribute ID doit être USINT, UINT, ou UDINT.

[b] Ce paramètre doit exister quand la cible de la demande se trouve dans un sous-réseau qui ne prend pas en charge l'adressage de message explicite au niveau de l'attribut. Si le sous-réseau ne prend pas en charge l'adressage au niveau de l'attribut et du membre, ce paramètre ne doit pas exister. Dans ce dernier cas, le type de données de Member ID/EX peut être SWORD, WORD ou DWORD, le bit le plus significatif étant le paramètre Extended Protocol Option et les bits restants le Member ID.

La structure du corps _ResponsePDU figure au Tableau 82.

Tableau 82 – Structure du corps _Member_ResponsePDU (Multiple Sequential Members)

Nom	Type de données	Description du paramètre
Nombre de membres	UINT	Nombre de membres bénéficiaires du service Ce champ est obligatoire.
Member ID (conditionnel)	UINT	Member ID du premier membre bénéficiaire du service Ce champ est nécessaire à certains services, voir la description individuelle des services. Ce champ est nécessaire si le champ Member Data existe.
Member Data (conditionnel)	Spécifique à un membre	Données du membre séquentiel multiple Ce champ est nécessaire à certains services, voir la description individuelle des services.

4.1.8.1.21.6 Demande de membre – format étendu (International String Selection – sélection de chaîne internationale)

Ce service retourne une seule chaîne de caractères internationaux à partir d'un attribut dont le type de données est STRINGI. Ce service peut nécessiter une langue donnée ou utiliser par défaut la langue courante activée comme le spécifie l'objet Identity.

Le protocole International String Selection est valide uniquement avec le service Get_Member.

La structure du corps _RequestPDU est indiquée dans le Tableau 83.

Tableau 83 – Structure du corps _RequestPDU (International String Selection)

Nom	Type de données	Description du paramètre
Identificateur d'attribut	USINT	Identifie l'attribut qui contient la chaîne internationale.
Member ID/ EX	WORD	Les 15 bis de plus faible poids identifient la langue à rechercher. Cette valeur est le Member ID (indice de matrice, commence à un) de la langue dans l'attribut Supported Language List (Attribute ID 12) de l'objet Identity. Option de protocole étendu EX = 1 (bit 15)
Extended Protocol ID (Identificateur de protocole étendu)	USINT	Sélectionne l'option de protocole Sélection de chaîne internationale (valeur = 2).

La structure du corps _ResponsePDU figure au Tableau 84.

Tableau 84 – Structure du corps _Member_ResponsePDU (International String Selection, sélection de chaîne internationale)

Nom	Type de données	Description du paramètre
Chaîne internationale	STRUCT de	Chaîne de caractères dans la langue demandée. Ce paramètre suit la définition d'une chaîne unique de caractères internationaux dans le type de données STRINGI (voir 4.2).
	USINT	Le champ language1 issu du type de données STRINGI.
	USINT	Le champ language2 issu du type de données STRINGI.
	USINT	Le champ language3 issu du type de données STRINGI.
	USINT	Structure de la chaîne de caractères, limitée aux valeurs 0xD0, 0xD5, 0xD9 et 0xDA.
	UINT	Ensemble de caractères de la chaîne internationale.
	OCTET_STRING	Chaîne internationale

4.1.8.1.22 Group_Sync

Le corps de Group_Sync_RequestPDU doit être tel que spécifié dans le Tableau 85.

Tableau 85 – Structure du corps de Group_Sync_RequestPDU

Nom	Type de données
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Group_Sync_ResponsePDU doit être tel que spécifié dans le Tableau 86.

Tableau 86 – Structure du corps de Group_Sync_ResponsePDU

Nom	Type de données	Sémantique des valeurs
IsSynchronized	BOOL	Indique si l'objet est synchronisé à la Base de temps PTP 0 = Not synchronized (Non synchronisé) 1 = Is synchronized (Est synchronisé).
Object Specific Data	Spécifique à un service d'objet/de classe	Spécifique à un service d'objet/de classe

4.1.8.2 Eléments de syntaxe spécifiques à l'objet Identity

4.1.8.2.1 Attributs

4.1.8.2.1.1 Attributs de classe

Le format des attributs de classe d'objets Identity doit être tel que spécifié dans le Tableau 87.

Tableau 87 – Attributs de classe d'objets Identity

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
1	Revision	UINT	Révision de cet objet. Valeur = 1
2	Max Instance	UINT	
6	Max ID Number of Class Attributes	UINT	
7	Max ID Number of Instance Attributes	UINT	

4.1.8.2.1.2 Attributs d'instance

Le format des attributs d'instance d'objet Identity est tel que spécifié dans le Tableau 88.

Tableau 88 – Attributs d'instance d'objet Identity

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
1	ID de vendeur	UINT	La valeur zéro ne doit pas être utilisée.
2	Device Type	UINT	Une énumération des plages de Device Type peut être consultée en 4.1.10.2.5.
3	Product Code	UINT	La classe zéro n'est pas valide.
4	Revision	STRUCT de	La valeur zéro ne doit être valide pour aucun des champs Major Revision et Minor Revision d'un produit conforme. Un produit endommagé peut retourner zéro pour indiquer une révision inconnue si son stockage de révision devient corrompu.

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
	Major Revision	USINT	L'attribut Major Revision (Révision majeure) doit être limité aux valeurs comprises entre 1 et 127 (7 bits). Le huitième bit est réservé et doit être zéro.
	Minor Revision	USINT	1 à 255
5	Status	WORD	Les définitions des bits sont décrites dans le Tableau 89.
6	Serial Number	UDINT	
7	Product Name	SHORT_STRING	Le nombre maximal de caractères de 8 bits dans cette chaîne doit être de 32. Chacun des caractères dans se trouver dans la plage 0x20 à 0x7E.
8	State	USINT	Présent état de l'appareil tel que représenté par le diagramme de transitions d'états: 0 = Nonexistent 1 = Device Self Testing 2 = Standby 3 = Operational 4 = Major Recoverable Fault 5 = Major Unrecoverable Fault 6 à 254 = Reserved 255 = Valeur par défaut [a]
9	Configuration Consistency Value	UINT	Le contenu identifie la configuration de l'appareil.
10	Heartbeat Interval	USINT	L'intervalle nominal entre les messages "heartbeat", en secondes.
11	Active Language	STRUCT de	Langue actuellement active pour l'appareil.
	Language1	USINT	Le champ language1 issu du type de données STRINGI.
	Language2	USINT	Le champ language2 issu du type de données STRINGI.
	Language3	USINT	Le champ language3 issu du type de données STRINGI.
12	Supported Language	MATRICE de STRUCT de	Liste des langues prises en charge par les chaînes de caractères du type de données STRINGI au sein de l'appareil
	Language1	USINT	Le champ language1 issu du type de données STRINGI.
	Language2	USINT	Le champ language2 issu du type de données STRINGI.
	Language3	USINT	Le champ language3 issu du type de données STRINGI.
13	International Product Name	STRINGI	Noms du produit dans diverses langues
14	Semaphore	STRUCT de	Access Synchronization Semaphore (Sémaphore de synchronisation d'accès)
	Vendor Number	UINT	Par défaut: Toutes les valeurs nulles
	Client Serial Number	UDINT	
	Semaphore Timer	ITIME	Plage valide du temporisateur 100 à 32 767
15	Assigned_Name	STRING1	Nom attribué par l'utilisation
16	Assigned_Description	STRING1	Description attribuée par l'utilisateur
17	Geographic_Location	STRING1	Emplacement attribué par l'utilisateur
18	Type15 Identity Info		Réservé aux appareils de Type 15.

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
[a] La valeur par défaut doit être utilisée dans la réponse Get_Attribute_All si l'attribut n'est pas mis en œuvre.			

Tableau 89 – Définitions de bit de l'objet Identity pour l'attribut d'instance de statut

Bit(s)	Appelé	Définition
0	Owned (Possédé)	1 pour TRUE, 0 pour FALSE
1		Réservé, mis à 0
2	Configured (Configuré)	1 pour TRUE, 0 pour FALSE
3		Réservé, mis à 0
4 à 7	Extended Device Status	Spécifique à un vendeur ou spécifié au Tableau 90
8	Minor Recoverable Fault (Défaut récupérable mineur)	1 pour TRUE, 0 pour FALSE
9	Minor Unrecoverable Fault (Défaut irrécupérable mineur)	1 pour TRUE, 0 pour FALSE
10	Major Recoverable Fault (Défaut récupérable majeur)	1 pour TRUE, 0 pour FALSE
11	Major Unrecoverable Fault (Défaut irrécupérable majeur)	1 pour TRUE, 0 pour FALSE
12 à 15	Extended Device Status 2	Réservés (doivent être mis à 0) ou spécifiques au vendeur

Les valeurs des bits de statut 4 à 7 doivent être telles que montrées dans le Tableau 90.

Tableau 90 – Valeurs par défaut pour le champ statut étendu de l'appareil (bits 4 à 7) de l'attribut d'instance de statut

Valeur	Signification
0	En autotest (mise sous tension) ou état inconnu
1	Mise à jour de firmware en cours
2	Défaut de communication (au moins une connexion perdue)
3	Déverrouillé, en attente de connexion, pas de connexion E/S établie
4	Mauvaise configuration non volatile
5	Défaut majeur (voir bits 10 et 11 pour la classification des défauts et les états des DEL) Un défaut mineur n'est pas un changement d'état.
6	Connecté, actif, au moins une connexion E/S en mode marche
7	Inactif (type mode de programme), au moins une connexion E/S établie, toutes en mode inactif
8 et 9	réservé, ne doit pas être utilisé
10 à 15	réservés aux états spécifiques à un vendeur

4.1.8.2.2 Services communs

4.1.8.2.2.1 Réponse Get_Attribute_All

Au **niveau de la classe**, l'ordre des attributs retournés dans le paramètre “Attribute Data” de la réponse Get_Attribute_All doit tel que défini dans le Tableau 91.

Tableau 91 – Données de réponse spécifiques à un service/objet de niveau classe pour Get_Attribute_All

Identificateur d'attribut	Type de données	Nom d'attribut	Valeur par défaut (si elle n'est pas mise en œuvre)
1	UINT	Revision	1
2	UINT	Max Instance	1
6	UINT	Max ID Number Class Attributes	0
7	UINT	Max ID Number Instance Attributes	0

Les valeurs par défaut doivent être insérées pour tous les attributs non pris en charge.

Au **niveau de l'instance**, l'ordre des attributs retournés dans le paramètre “Attribute Data” de la réponse Get_Attribute_All doit tel que défini dans le Tableau 92.

Tableau 92 – Données de réponse spécifiques à un service/objet de niveau instance pour Get_Attribute_All

Identificateur d'attribut	Type de données	Nom d'attribut	Valeur par défaut (si elle n'est pas mise en œuvre)
1	UINT	Vendor ID	
2	UINT	Device Type	
3	UINT	Product Code	
4	STRUCT of:	Revision	
	USINT	Major Revision	
	USINT	Minor Revision	
5	WORD	Status	
6	UDINT	Serial Number	
7	SHORT STRING	Product Name	
8	USINT	State	255
9	UINT	Configuration Consistency Value	0
10	USINT	Heartbeat Interval	0

Les valeurs par défaut doivent être insérées pour tous les attributs non pris en charge.

Si le service Get_Attribute_All retourne des données après Heartbeat Interval et que la révision de l'objet Identity est 1, les données postérieures à Heartbeat Interval (octet n+4) doivent être ignorées (étant donné que les éditions précédentes de la norme comportaient du contenu non analysable passé ce point).

4.1.8.2.2.2 Service Reset

Le service commun Reset doit avoir le paramètre spécifique à un objet spécifié dans le Tableau 93.

Tableau 93 – Paramètre spécifique à un objet pour Reset

Nom	Type	Sémantique des valeurs
Type	USINT	Type de Réinitialisation Voir Tableau 94.

Tableau 94 – Valeurs de paramètre du service Reset

Valeur	Type de Réinitialisation
0	Emuler aussi étroitement que possible la puissance en cycle sur l'élément que l'objet Identity représente. Cette valeur doit être la valeur par défaut si ce paramètre est omis.
1	Restituer aussi fidèlement que possible la configuration par défaut d'atelier, puis émuler la puissance en cycle aussi fidèlement que possible.
2	Restituer aussi fidèlement que possible la configuration "out-of-box" avec l'exception des paramètres de liaison de communication, et émuler la puissance en cycle aussi fidèlement que possible. Les paramètres de liaison de communication qui sont à préserver sont définis par chaque type de réseau. Voir le service Reset (réinitialisation) de l'objet ou des objets de liaison spécifiques au réseau pour avoir des informations circonstanciées;
3 à 99	Réservé.
100 à 199	Vendor specific (Spécifique à un vendeur)
200 à 255	Réservé

4.1.8.3 Eléments de syntaxe spécifiques à l'objet Message Router

4.1.8.3.1 Attributs

4.1.8.3.1.1 Attributs de classe

Les attributs de classe d'objets Message Router doivent être tels que spécifiés dans le Tableau 95.

Tableau 95 – Attributs de classe d'objets Message Router

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
1	Revision	UINT	Révision de cet objet. Valeur = 1
4	Optional attribute list	STRUCT of	
	Number of attributes	UINT	
	Optional attributes	ARRAY of UINT	
5	Optional service list	STRUCT of	
	Number services	UINT	
	Optional services	ARRAY of UINT	
6	Max ID Number of Class Attributes	UINT	
7	Max ID Number of Instance Attributes	UINT	

4.1.8.3.1.2 Attributs d'instance

Le format des attributs d'instance d'objet Message Router est tel que spécifié dans le Tableau 96.

Tableau 96 – Attributs d'instance d'objet Message Router

Identificateur d'attribut	Nom	Type de données
1	Object_List	STRUCT of
	Number	UINT
	Classes	ARRAY of UINT

Identificateur d'attribut	Nom	Type de données
2	Number available	UINT

4.1.8.3.2 Services communs

Réponse Get_Attribute_All

Au **niveau classe**, l'ordre des attributs retournés dans le paramètre "Attribute Data" de la réponse Get_Attribute_All doit tel que spécifié dans le Tableau 97. Les valeurs par défaut pour tous les attributs de classe non pris en charge doivent être telles que montrées dans le tableau en question.

Tableau 97 – Données de réponse spécifiques à un service/objet de niveau classe pour Get_Attribute_All

ID d'attribut de classe	Nom d'attribut, description et valeur par défaut
1	Revision valeur par défaut = 1
4	Optional attribute list, nombre d'attributs, valeur par défaut = 0
5	Optional service list, nombre de services, valeur par défaut = 0
6	Max ID, nombre d'attributs de classe, valeur par défaut = 0
7	Max ID, nombre d'attributs d'instance, valeur par défaut = 0

Les valeurs par défaut pour tous les attributs d'instance non pris en charge doivent être telles que montrées dans le Tableau 98.

Tableau 98 – Données de réponse spécifiques à un service/objet de niveau instance pour Get_Attribute_All

ID d'attribut d'instance	Nom d'attribut, description et valeur par défaut
1	Object list number, nombre de classes prises en charge, valeur par défaut = 0
2	Number available, nombre maximal de connexions, valeur par défaut = 0
3	Toujours = 0x0000

4.1.8.3.3 Services spécifiques à un objet

Symbolic_Translation

Le corps de Symbolic_Translation_RequestPDU doit être tel que spécifié dans le Tableau 99.

Tableau 99 – Structure du corps de Symbolic_Translation_RequestPDU

Nom	Type de données
Symbolic Address	Padded EPATH

Le corps de Symbolic_Translation_RequestPDU doit être tel que spécifié dans le Tableau 100.

Tableau 100 – Structure du corps de Symbolic_Translation_RequestPDU

Nom	Type de données
Logical Address	Padded EPATH

La réponse de service peut retourner un statut issu de la liste de codes de statut dans le Tableau 101.

Tableau 101 – Statut spécifique à un objet pour le service Symbolic_Translation

Code de statut général	Code de statut étendu	Nom de l'erreur	Description de statut
0x20	0x00	Chemin symbolique inconnu	Le nœud de traitement ne reconnaît pas le chemin symbolique
0x20	0x01	Destination du chemin symbolique non attribuée	Bien que le chemin symbolique soit reconnu, il n'est pas associé actuellement à un équivalent de chemin logique.
0x20	0x02	Symbolic Path segment error (Erreur de segment de chemin symbolique)	L'identificateur de symbole ou la syntaxe du segment de symbole n'a pas été compris par le nœud de traitement.

4.1.8.4 Eléments de syntaxe spécifiques à un objet Assembly

4.1.8.4.1 Attributs

4.1.8.4.1.1 Attributs de classe

Le format des attributs de classe d'objets Assembly doit être tel que spécifié dans le Tableau 102.

Tableau 102 – Attributs de classe d'objets Assembly

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
1	Revision	UINT	Révision de cet objet. Valeur = 2
2	Max Instance	UINT	

4.1.8.4.1.2 Attributs d'instance

Le format des attributs d'instance d'objet Assembly est tel que spécifié dans le Tableau 103.

Tableau 103 – Attributs d'instance d'objet Assembly

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
1	Number of Members in List	UINT	
2	Member List	ARRAY of STRUCT	Cet attribut à un type de données complexe.
	Member Data Size	UINT	Taille en bits
	Member Path Size	UINT	Taille en octets (0 = Empty Path (Chemin vide))
	Member Path	Packed EPATH	Chemin "condensé" vers les données de membre. Voir 4.1.8.9 pour la définition de Path
3	Data	ARRAY of SWORD	Contient toutes les données de membre condensées en une matrice. Ces données peuvent contenir plusieurs types de données différents. Pour l'efficacité, il vaut mieux maintenir ce mot de données aligné en le condensant sur les frontières des mots et en ajoutant un bourrage selon les besoins. Ceci peut être accompli en utilisant des "chemins vides" (Member Path Size = 0)

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
4	Size	UINT	

4.1.8.4.1.3 Plages des Assembly Instance ID

Les valeurs de Assembly Instance ID doivent être celles qui figurent au Tableau 104. Assembly Instance ID = 0x00 est réservé et ne doit pas être utilisé.

Tableau 104 – Plages des Assembly Instance ID

Plage (en hex)	Plage (en décimal)	Signification	Quantité
01 à 63	00 à 99	Ouvert[a]	99
64 à C7	100 à 199	Spécifique à un vendeur[b]	100
C8 à 2FF	200 à 767	Ouvert[a]	568
300 à 4FF	768 à 1 279	Spécifique à un vendeur[b]	512
500 à FFFFF	1 280 à 1 048 575	Ouvert[a]	1 047 296
100000 à FFFFFFFF	1 048 576 à 4 294 967 295	Réservé [c]	4 293 918 720

[a] Assembly statiques, définis dans les profils d'appareil

[b] Assembly statiques et Assembly dynamiques

[c] Les plages réservées peuvent être définies par ODVA, Inc.

4.1.8.5 Eléments de syntaxe spécifiques à un objet Acknowledge Handler

4.1.8.5.1 Attributs

4.1.8.5.1.1 Attributs de classe

Le format des attributs de classe d'objets Acknowledge Handler doit être tel que spécifié dans le Tableau 105.

Tableau 105 – Attributs de classe d'objets Acknowledge Handler

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
1	Revision	UINT	Révision de cet objet. Valeur = 1
4	Optional attribute list	STRUCT of	
	Number of attributes	UINT	
	Optional attributes	ARRAY of UINT	
5	Optional service list	STRUCT of	
	Number services	UINT	
	Optional services	ARRAY of UINT	
6	Max ID Number of Class Attributes	UINT	
7	Max ID Number of Instance Attributes	UINT	

4.1.8.5.1.2 Attributs d'instance

Le format des attributs d'instance d'objet Acknowledge Handler est tel que spécifié dans le Tableau 106.

Tableau 106 – Attributs d'instance d'objet Acknowledge Handler

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
1	Acknowledge Timer	UINT	Plage 1 ms à 65 535 ms (0 non valide) par défaut = 16
2	Retry Limit	USINT	Plage 0 à 255 Valeur par
3	COS Producing Connection Instance	UINT	Connection Instance ID
4	Ack List Size	SWORD	0 = Dynamique > 0 = Nombre max de membres
5	Ack List	STRUCT of	
		SWORD	Nombre de membres dans la matrice
		ARRAY of UINT	Liste d'identificateurs d'instance de connexion
6	Data with Ack Path List Size	SWORD	0 = Dynamique > 0 = Nombre max de membres
7	Data with Ack Path List	STRUCT of	
		SWORD	Nombre de membres dans la matrice
		ARRAY of	
		UINT	Connection Instance ID
		USINT	Longueur de chemin
		Padded EPATH	Chemin

4.1.8.5.2 Services spécifiques à un objet

4.1.8.5.2.1 Add_AckData_Path

Le corps de l'Add_AckData_Path_RequestPDU doit être tel que spécifié dans le Tableau 107.

Tableau 107 – Structure du corps de Add_AckData_Path_RequestPDU

Nom	Type de données
Connection Instance ID	UINT
Consumer Path Size	USINT
Consumer Path	Padded EPATH

Le corps de l'Add_AckData_Path_ResponsePDU est vide.

4.1.8.5.2.2 Remove_AckData_Path

Le corps de Remove_AckData_Path_RequestPDU doit être tel que spécifié dans le Tableau 108.

Tableau 108 – Structure du corps de Remove_AckData_Path_RequestPDU

Nom	Type de données
Connection Instance ID	UINT

Le corps de Remove_AckData_Path_ResponsePDU est vide.

4.1.8.6 Eléments de syntaxe spécifiques à un objet Time Sync

4.1.8.6.1 Attributs

4.1.8.6.1.1 Attributs de classe

Le format des attributs de classe d'objet Time Sync doit être tel que spécifié dans le Tableau 109.

Tableau 109 – Attributs de classe d'objet Time Sync

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
1	Revision	UINT	Révision de cet objet. Valeur = 3

4.1.8.6.1.2 Attributs d'instance

Le format des attributs d'instance d'objet Time Sync doit être tel que spécifié dans le Tableau 110.

Tableau 110 – Attributs d'instance d'objet Time Sync

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
1	PTPEnable	BOOL	PTP activé pour cet appareil: 0 = PTP désactivé 1 = PTP activé
2	IsSynchronized	BOOL	Horloge locale synchronisée avec l'horloge mère de référence: 0 = horloge locale non synchronisée 1 = horloge locale synchronisée
3	SystemTimeMicroseconds	ULINT	Valeur courante de system_time en microsecondes
4	SystemTimeNanoseconds	ULINT	Valeur courante de system_time en nanosecondes
5	OffsetFromMaster	LINT	Décalage entre l'horloge locale et l'horloge mère en nanosecondes
6	MaxOffsetFromMaster	LINT	Décalage maximal entre l'horloge locale et l'horloge mère en nanosecondes
7	MeanPathDelayToMaster	LINT	Retard de chemin moyen par rapport à la mère en nanosecondes
8	GrandMasterClockInfo	STRUCT of	Informations d'horloge grand-mère
	ClockIdentity	USINT[8]	Identificateur unique de l'horloge. Voir Tableau 111 pour le codage de ClockIdentity.
	ClockClass	UINT	Classe de la qualité d'horloge. Le Tableau 112 montre les valeurs les plus susceptibles d'être utilisées par l'objet Time Sync. Voir la CEI 61588:2009 pour une liste complète de valeurs.

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
	TimeAccuracy	UINT	Exactitude absolue prévue de l'horloge par rapport à l'époque PTP. Voir le Tableau 113 pour les valeurs de TimeAccuracy.
	OffsetScaledLogVariance	UINT	Mesure des propriétés intrinsèques de stabilité de l'horloge
	CurrentUtcOffset	UINT	Décalage TUC courant, en secondes, par rapport au Temps atomique international (TAI) de l'horloge. Au 1er janvier 2006 à 00:00:00 TUC, le décalage était de 33 secondes.
	TimePropertyFlags	WORD	Pour la valeur de TimePropertyFlags, voir le Tableau 114.
	TimeSource	UINT	Pour les valeurs possibles de TimeSource, voir le Tableau 115.
	Priority1	UINT	Priorité relative de l'horloge grand-mère par rapport aux autres horloges dans le système. Voir l'attribut 16 Priority1
	Priority2	UINT	Priorité relative de l'horloge grand-mère par rapport aux autres horloges dans le système. Voir l'attribut 17 Priority2
9	ParentClockInfo	STRUCT of	Informations d'horloge parente
	ClockIdentity	USINT[8]	Identificateur unique de l'horloge. Voir Tableau 111 pour le codage de ClockIdentity.
	PortNumber	UINT	Numéro de l'identité de port
	ObservedOffsetScaledLog Variance	UINT	Mesure estimée de la variance de l'horloge parente telle qu'observée par l'horloge esclave
	ObservedPhaseChangeRate	UDINT	Mesure estimée de la dérive de l'horloge parente telle qu'observée par l'horloge esclave
10	LocalClockInfo	STRUCT of	Informations d'horloge locale
	ClockIdentity	USINT[8]	Identificateur unique de l'horloge. Voir le Tableau 111 pour le codage de ClockIdentity.
	ClockClass	UINT	Classe de la qualité d'horloge. Le Tableau 112 montre les valeurs les plus susceptibles d'être utilisées par l'objet Time Sync. Voir la CEI 61588:2009 pour une liste complète de valeurs.
	Time Accuracy	UINT	Exactitude absolue prévue de l'horloge par rapport à l'époque PTP. Voir le Tableau 113 pour les valeurs de TimeAccuracy.
	OffsetScaledLogVariance	UINT	Mesure des propriétés intrinsèques de stabilité de l'horloge
	CurrentUtcOffset	UINT	Décalage TUC courant, en secondes, par rapport au Temps atomique international (TAI) de l'horloge. Au 1er janvier 2006 à 00:00:00 TUC, le décalage était de 33 secondes.
	TimePropertyFlags	WORD	Pour la valeur de TimePropertyFlags, voir le Tableau 114.
	TimeSource	UINT	Pour les valeurs possibles de TimeSource, voir le Tableau 115.
11	NumberOfPorts	UINT	Nombre de ports PTP sur l'appareil
12	PortStateInfo	STRUCT of	Informations d'état de port
	NumberOfPorts	UINT	Nombre de ports PTP sur l'appareil
		ARRAY of STRUCT	
	PortNumber	UINT	Numéro d'identité du port

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
	PortState	UINT	Etat courant du port PTP: 1 = Initializing (Initialisation) 2 = Faulty (Défectueux) 3 = Disabled (Désactivé) 4 = Listening (A l'écoute) 5 = Pre_Master 6 = Master (Maître) 7 = Passive (Passif) 8 = Uncalibrated (Non étalonné) 9 = Slave (Esclave) All other (Tout autre) = Reserved (Réservé)
13	PortEnableCfg	STRUCT of	Configuration d'activation de port
	NumberOfPorts	UINT	Nombre de ports PTP sur l'appareil
		ARRAY of STRUCT	
	PortNumber	UINT	Numéro d'identité du port
	PortEnable	UINT	PortEnable a les valeurs suivantes (la valeur par défaut est "activé"): 1 = port activé 0 = port désactivé
14	PortLogAnnounceIntervalCfg	STRUCT of	Configuration de l'intervalle entre annonces de journal de port
	NumberOfPorts	UINT	Nombre de ports PTP sur l'appareil
		ARRAY of STRUCT	Port log announce interval of each port
	PortNumber	UINT	Numéro d'identité du port
	PortLogAnnounceInterval	UINT	Intervalle d'annonces PTP entre des messages "Announce" successifs émis par l'horloge mère
15	PortLogSyncIntervalCfg	STRUCT of	Configuration de l'intervalle de synchronisation de journal de port
	NumberOfPorts	UINT	Nombre de ports PTP sur l'appareil
		ARRAY of STRUCT	
	PortNumber	UINT	Numéro d'identité du port
	PortLogSyncInterval	UINT	Intervalle de sync PTP entre des messages "Sync" successifs émis par l'horloge mère
16	Priority1	USINT	Notation Best Master de cette horloge; annule et remplace la qualité d'horloge (classe, exactitude et variance). Les valeurs sont comprises entre 0 et 255, 0 correspondant à la priorité la plus élevée
17	Priority2	USINT	Notation Best Master de cette horloge après que la qualité d'horloge (classe, exactitude et variance) a été évaluée; annule et remplace le tie-breaker. Les valeurs sont comprises entre 0 et 255, 0 correspondant à la priorité la plus élevée
18	DomainNumber	USINT	Domaine d'horloge PTP
19	ClockType	WORD	Pour les valeurs de ClockType, voir le Tableau 116
20	ManufactureIdentity	USINT[4]	Identité de fabricant de l'horloge. Les 3 premiers octets spécifient l'OUI (Organization Unique ID "identificateur propre à une organisation") de l'IEEE pour le fabricant. Le dernier octet est réservé.
21	ProductDescription	STRUCT of	Description de produit de l'appareil qui contient l'horloge

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
	Size	UDINT	Taille de la description du produit (nombre total d'octets pour le champ Description) [a]
	Description	USINT[Size]	Description du produit au format Unicode UTF-8 avec 64 symboles au maximum
22	RevisionData	STRUCT of	Données de révision de l'appareil qui contient l'horloge
	Size	UDINT	Taille des données de révision (nombre total d'octets pour le champ Revision) [a]
	Revision	USINT[Size]	Données de révision au format Unicode UTF-8 avec 32 symboles au maximum
23	UserDescription	STRUCT of	Description d'utilisateur de l'appareil qui contient l'horloge
	Size	UDINT	Taille de la description de l'utilisateur (nombre total d'octets pour le champ Description) [a]
	Description	USINT[Size]	Description d'utilisateur au format Unicode UTF-8 avec 128 symboles au maximum
24	PortProfileIdentityInfo	STRUCT of	Les informations d'identité de profil de port doivent être:
	NumberOfPorts	UINT	Nombre de ports PTP sur l'appareil
		ARRAY of STRUCT	
	PortNumber	UINT	Numéro d'identité du port
	PortProfileIdentity	USINT[8]	Identité de profil de port. L'identificateur de profil est contenu dans les six premiers octets de la matrice. Les deux derniers octets doivent être mis à zéro. Pour un profil CPF 2, l'identificateur de profil doit être: 00-21-6C-00-01-00
25	PortPhysicalAddressInfo	STRUCT of	Informations d'adresse physique de port
	NumberOfPorts	UINT	Nombre de ports PTP sur l'appareil
		ARRAY of STRUCT	
	PortNumber	UINT	Numéro d'identité du port
	PhysicalProtocol	USINT[16]	Protocole physique, exprimé en caractères ASCII. Le nombre maximal de caractères est 16. Les éléments inutilisés de la matrice doivent avoir pour valeur zéro. Voir la Tableau 117 qui présente une liste des valeurs recommandées.
	SizeOfAddress	UINT	Taille de PortPhysicalAddress
	PortPhysicallAddress	USINT[16]	Adresse du port physique, exprimée en caractères ASCII. Le nombre maximal de caractères est 16. Les éléments inutilisés de la matrice doivent avoir pour valeur zéro. Voir la Tableau 117 qui présente une liste des valeurs recommandées.
26	PortProtocolAddressInfo	STRUCT of	Informations d'adresse de protocole de port
	NumberOfPorts	UINT	Nombre de ports PTP sur l'appareil
		ARRAY of STRUCT	
	PortNumber	UINT	Numéro d'identité du port

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
	NetworkProtocol	UINT	Valeurs de NetworkProtocol 1 = UDP/IPv4 (p. ex. CP 2/2) 2 = UDP/IPv6 3 = ISO/IEC 8802-3 4 = CP 2/3 5 = CP 2/1 0xFFFE = protocole local ou inconnu tous les autres = réservés
	SizeOfAddress	UINT	Taille de PortProtocolAddress
	PortProtocolAddress	USINT[16]	Adresse de protocole de port (par exemple, adresse IP) Le nombre maximal de caractères est 16. Les éléments inutilisés de la matrice doivent avoir pour valeur zéro.
27	StepRemoved	UINT	Nombre de chemins de communication traversés entre l'horloge locale et l'horloge grand-mère
28	SytemTimeAndOffset	STRUCT of	Heure système et décalage
	SystemTime	ULINT	Heure système en microsecondes
	SystemOffset	ULINT	Décalage de la valeur d'horloge locale

[a] Le nombre de symboles de la description doit être converti en octets.

Tableau 111 – Codage de ClockIdentity pour différentes mises en œuvre de réseau

Réseau	Octet 0	Octet 1	Octet 2	Octet 3	Octet 4	Octet 5	Octet 6	Octet 7
CP 2/1	0xFF	0x02	Vendor ID [a]		Serial Number [a]			
CP 2/2	MAC Address [b]			0xFF	0xFE	MAC Address [b]		
CP 2/3	0xFF	0x01	Vendor ID [a]		Serial Number [a]			
Local ou fermé	0xFF	0xFF	Vendor ID [a]		Serial Number [a]			
Tous les autres	Voir la CEI 61588:2009.							

a L'octet le plus significatif du Vendor ID et du Serial Number est attribué à l'octet le plus significatif de ClockIdentity. Par exemple, si le Vendor ID est "0809"et le Serial Number "03040506", la ClockIdentity pour CP 2/3 est "FF 01 08 09 03 04 05 06".

b L'octet le plus significatif de MAC Address est attribué à l'octet le plus significatif de ClockIdentity. Par exemple si l'adresse MAC est "00 01 02 03 04 05" la ClockIdentity est "00 01 02 FF FE 03 04 05".

Tableau 112 – Valeurs de ClockClass

ClockClass	Valeur
Référence primaire	6
Référence primaire (Hold)	7
Référence dégradée A (Maître seulement)	52
Référence dégradée B (Maître / Esclave)	187
Valeur par défaut	248
Esclave seulement	255
Voir CEI 61588:2009	tous les autres

Tableau 113 – Valeurs de TimeAccuracy

Exactitude d'horloge	Valeur
Réservé	0x00 à 0xFF
± 25 ns	0x20
±100 ns	0x21
±250 ns	0x22
±1 µs	0x23
±2,5 µs	0x24
±10 µs	0x25
±25 µs	0x26
±100 µs	0x27
±250 µs	0x28
±1 ms	0x29
±2,5 ms	0x2A
±10 ms	0x2B
±25 ms	0x2C
±100 ms	0x2D
±250 ms	0x2E
±1 s	0x2F
±10 s	0x30
> ± 10 s	0x31
Réservé	0x32 à 0x7F
Pour utilisation par des profils PTP alternatifs	0x80 à 0xFD
Inconnu	0xFE
Réservé	0xFF

Tableau 114 – Valeurs de bit de TimePropertyFlags

TimePropertyFlags	Indice de bit
Leap indicator 61	0
Leap indicator 59	1
Current UTC offset valid	2
PTP timescale	3
Time traceable	4
Frequency traceable	5

Tableau 115 – Valeurs de TimeSource

TimeSource	Valeur
ATOMIC CLOCK	0x10
GPS	0x20
TERRESTRIAL RADIO	0x30
PTP	0x40
NTP	0x50
HAND SET	0x60
OTHER	0x90
INTERNAL OSCILLATOR	0xA0

Pour utilisation par des profils PTP alternatifs	0xF0 à 0xFE
Réservé	0xFF

Tableau 116 – Types d'horloge

Indice de bit	Type d'horloge configuré
0	Réservé [a]
1	Réservé [a]
2	Réservé [a]
3	Nœud de gestion
4	Horloge transparente de bout en bout
5	Réservé [a]
6	Horloge frontière
7	Horloge ordinaire
8	Esclave seulement
9 à 15	Réservé [a]

[a] Les bits réservés doivent être mis à 0.

Tableau 117 – Protocole réseau pour la mise en correspondance PortPhysicalAddressInfo

Protocole réseau	PhysicalProtocol	PortPhysicalAddress
UDP/IPv4 (par exemple, CP 2/2)	"IEEE 802.3"	Utiliser l'adresse MAC
UDP/IPv6	"IEEE 802.3"	Utiliser l'adresse MAC
IEEE 802.3	"IEEE 802.3"	Utiliser l'adresse MAC
CP 2/1	"ControlNet"	Utiliser l'ID MAC
CP 2/3	"DeviceNet"	Utiliser l'ID MAC
Protocole local ou inconnu	Spécifique à un vendeur	Utiliser un élément spécifique au vendeur

4.1.8.6.2 Services communs

Réponse Get_Attribute_All

Au niveau classe, la réponse Get_Attributes_All doit contenir les attributs de classe dans l'ordre numérique, jusqu'au dernier attribut mis en œuvre. Tous les attributs non mis en œuvre dans la réponse doivent utiliser les valeurs d'attribut par défaut.

4.1.8.7 Eléments de syntaxe spécifiques à un objet Parameter

4.1.8.7.1 Attributs

4.1.8.7.1.1 Attributs de classe

Le format des attributs de classe d'objets Parameter doit être tel que spécifié dans le Tableau 118.

Tableau 118 – Attributs de classe d'objets Parameter

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
1	Revision	UINT	Révision de cet objet. Valeur = 1
2	Max Instance	WORD	Le numéro d'instance le plus élevé d'un objet créé à ce niveau de hiérarchie de classe
8	Parameter Class Descriptor	UINT	Voir Tableau 119
9	Configuration Assembly Instance	UINT	Cet attribut doit être mis à zéro si l'assemblage de configuration n'est pas pris en charge.
10	Native Language	UINT	0 = Anglais 1 = Français 2 = Espagnol 3 = Italien 4 = Allemand 5 = Japonais 6 = Portugais 7 = Chinois mandarin

L'attribut Parameter Class Descriptor (descripteur de classe de paramètre) contient les bits pour décrire des caractéristiques de paramètre. Les bits pris en charge sont spécifiés dans le Tableau 119.

Tableau 119 – Valeurs de bit de Parameter Class Descriptor

Bit	Définition
0	Prend en charge les instances de Parameter. 1 = les instances individuelles de Parameter SONT prises en charge. 0 = AUCUNE instance individuelle de Parameter n'est prise en charge. Seule instance d'assemblage de configuration utilisée
1	Prend en charge les attributs complets (Full) 1 = Tous les attributs complets de Parameter SONT pris en charge. 0 = Seuls les attributs réduits d'instance de Parameter sont pris en charge.
2	Doit faire la commande sauvegarde de stockage non volatil 1 = Doit exécuter la commande de sauvegarde de stockage non volatil. Le service Save de la classe de paramètre est utilisé pour sauvegarder des valeurs de paramètre d'instance. 0 = Peut ne pas exécuter la commande sauvegarde de stockage non volatil.
3	Les paramètres sont stockés dans le stockage non volatil. 1 = Tous les paramètres complets sont stockés dans un stockage non volatil 0 = Des paramètres ne sont pas stockés dans un stockage non volatil

4.1.8.7.1.2 Attributs d'instance

Le format des attributs d'instance d'objet Parameter est tel que spécifié dans le Tableau 120.

Tableau 120 – Attributs d'instance d'objet Parameter

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
1	Parameter Value	Type de données spécifié dans Descriptor, Data Type et Data Size	
2	Link Path Size	USINT	Nombre d'octets
3	Link Path	Packed EPATH	Le Link Path (chemin de liaison) est limité à 255 octets
4	Descriptor	WORD	Voir Tableau 121
5	Data Type	USINT	Voir 5.2.2
6	Data Size	USINT	
7	Parameter Name String	SHORT_STRING	Le nombre maximal de caractères est 16.
8	Units String	SHORT_STRING	Le nombre maximal de caractères est 4.
9	Help String	SHORT_STRING	Le nombre maximal de caractères est 64.
10	Minimum Value	Data Type	Voir Tableau 122
11	Maximum Value	Data Type	Voir Tableau 122
12	Default Value	Data Type	Valeur effective à laquelle il convient de régler le paramètre lorsque l'utilisateur souhaite la valeur par défaut pour le paramètre
13	Scaling Multiplier	UINT	Voir Tableau 123
14	Scaling Divisor	UINT	Voir Tableau 123
15	Scaling Base	UINT	Voir Tableau 123
16	Scaling Offset	INT	Voir Tableau 123
17	Multiplier Link	UINT	Voir Tableau 124
18	Divisor Link	UINT	Voir Tableau 124
19	Base Link	UINT	Voir Tableau 124
20	Offset Link	UINT	Voir Tableau 124
21	Decimal Precision	USINT	Spécifie le nombre de décimales à utiliser pour afficher la valeur d'étude mise à l'échelle. Utilisé également pour déterminer la valeur d'incrément effective faisant que le fait d'incrémenter une valeur entraîne un changement de la valeur d'étude mise à l'échelle avec cette précision.
22	International Parameter Name	STRINGI	Chaîne lisible par l'homme qui représente le nom du paramètre
23	International Engineering Units	STRINGI	Unités d'étude associées au paramètre
24	International Help String	STRINGI	Chaîne lisible par l'homme qui représente la chaîne d'aide

Tableau 121 – Sémantique de l'attribut Descriptor Instance

Bit	Définition	Signification
0	Prend en charge le chemin réglable.	Indique que le chemin de liaison peut être établi.
1	Prend en charge Get_Enum_String.	Indique que les chaînes énumérées peuvent être lues avec le service Get_Enum_String. Si le bit descripteur 8 est mis, des valeurs

Bit	Définition	Signification
		énumérées situées en dehors de la plage min/max peuvent exister.
2	Prend en charge la mise à l'échelle.	Indique qu'il convient que le facteur d'échelle soit implémenté de manière à présenter la valeur à l'utilisateur en unités d'étude.
3	Prend en charge les liaisons de mise à l'échelle.	Indique que les valeurs pour le facteur d'échelle peuvent être récupérées d'autres paramètres. Si le bit descripteur "Supports Scaling" (Prend en charge la mise à l'échelle) est également mis et la valeur de liaison de mise à l'échelle est zéro, la valeur d'échelle doit être utilisée. Si le bit descripteur "Supports Scaling" (Prend en charge la mise à l'échelle) est également mis et la valeur de liaison de mise à l'échelle n'est pas zéro, la valeur de liaison de mise à l'échelle doit être utilisée.
4	Paramètre en lecture seule	Indique que l'attribut Parameter Value peut seulement être lu, et pas mis.
5	Paramètre de surveillance	Indique que l'attribut Parameter Value est mis à jour en temps réel par l'appareil.
6	Prend en charge la mise à l'échelle de précision étendue.	Indique que le facteur d'échelle de précision étendue doit être implémenté de manière à présenter la valeur à l'utilisateur en unités d'étude.
7	Prend en charge des chaînes énumérées non consécutives.	Désuet
8	Autorise l'énumération et aussi les valeurs de la plage min/max.	Valide uniquement pour les types de données integer (entier). Les valeurs énumérées et les valeurs de la plage min/max sont valides. Les valeurs énumérées peuvent se situer à l'intérieur ou à l'extérieur de la plage min/max. Si une valeur énumérée se situe à l'intérieur de cette plage, la chaîne énumérée prévaut sur la valeur.
9	Paramètre non affiché	L'outil de configuration ne doit pas afficher ce paramètre.
10	Référence de paramètre indirecte	La valeur du paramètre représente le numéro d'instance d'un autre paramètre. L'instance du paramètre référencé ne doit pas être une référence indirecte supplémentaire. Une valeur de paramètre égale à 0 doit indiquer l'absence de référence. Le type de données pour ce paramètre doit être USINT, UINT ou UDINT.
11	Non adressable	Le paramètre n'est adressable directement à partir du réseau par aucun des services Set_Attribute et Get_Attribute.
12	Save (Sauvegarder) pris en charge.	Ce bit, lorsqu'il est mis, indique que ce paramètre peut être individuellement sauvegardé en envoyant le service Save à cette instance de paramètre.
13	Apply (Appliquer) pris en charge.	Ce bit, lorsqu'il est mis, indique que l'exécution du service Apply sur cette instance de paramètre est requise avant que les changements d'attribut n'altèrent le comportement de l'instance.
14	Paramètre en écriture seule	Indique que l'attribut Parameter Value peut seulement être mis, et pas lu.
15	Réservé	Il doit être mis à zéro.

Tableau 122 – Sémantique de Minimum Value et de Maximum Value

Type de données	Description et sémantique	Sémantique de Minimum Value	Sémantique de Maximum Value	Requis/Facultatif/Interdit
OCTET	Bit String – longueur de 8 bits	Le bit de poids le plus faible qui est mis dans la valeur minimale est le bit valide de poids le plus faible devant être énuméré. Le bit de poids le plus fort qui est mis dans la valeur maximale est le bit valide de poids le plus fort devant être énuméré. Une valeur minimale égale à zéro signifie que le bit 0 est le bit valide de poids le plus faible devant être énuméré. Une valeur maximale égale à zéro n'est pas valide. Exemple 1 (type OCTET): Valeur min = 0b00000001 Valeur max = 0b00001000 Les bits 0 à 3 sont énumérés. Exemple 2 (type WORD): Valeur min = 0b0000000000000001 Valeur max = 0b0000000000001011		Facultatif
WORD	Bit String – longueur de 16 bits			
DWORD	Bit String – longueur de 32 bits			
LWORD	Bit String – longueur de 64 bits			

Type de données	Description et sémantique	Sémantique de Minimum Value	Sémantique de Maximum Value	Requis/Facultatif/Interdit
		Les bits 0 à 3 sont énumérés, les bits 0 et 1 de valeur max sont ignorés, le bit 3 de valeur min est ignoré. Exemple 3 (type DWORD): Valeur min = 0b00000000000000000000000000000000 Valeur max = 0b00000000000000000000000000001000 Les bits 0 à 3 sont énumérés, la valeur min 0 sous-entend le bit 0.		
STRING[a]	String (indicateur de longueur 2 octets, 1 octet par caractère)	Longueur de chaîne minimale	Longueur de chaîne maximale	Requis
STRING2[a]	String (indicateur de longueur 2 octets, 2 octets par caractère)	Longueur de chaîne minimale	Longueur de chaîne maximale	Requis
STRINGN[a]	String (indicateur de longueur 2 octets, N octets par caractère)	Longueur de chaîne minimale	Longueur de chaîne maximale	Requis
SHORT_STRING[a]	Character string (indicateur de longueur 1 octet, caractères de 1 octet)	Longueur de chaîne minimale	Longueur de chaîne maximale	Requis
STRINGI[a]	Chaîne de caractères internationaux	Longueur minimale (en caractères) du composant "stringcontents" de la séquence implicite de la chaîne	Longueur maximale (en caractères) du composant "stringcontents" de la séquence implicite de la chaîne	Requis
EPATH[a]	Path (chemin) codé	Longueur de chaîne minimale	Longueur de chaîne maximale	Facultatif
ENGUNIT	Unités d'étude	La valeur minimale pour ce type de données n'est pas définie et ne doit pas être spécifiée dans fichier EDS ou l'instance d'objet paramètre.	La valeur maximale pour ce type de données n'est pas définie et ne doit pas être spécifiée dans fichier EDS ou l'instance d'objet paramètre.	Facultatif
Tous les autres types de données		La valeur numérique minimale qui peut être attribuée à la valeur de donnée	La valeur numérique maximale qui peut être attribuée à la valeur de donnée	Facultatif[b]

[a] Les types de données STRING, STRING2, STRINGN, SHORT_STRING, STRINGI et EPATH n'ont pas de spécification de valeur minimale ou maximale. Les champs de valeur minimale et maximale sont utilisés pour présenter les longueurs maximale et minimale de chemin ou de chaîne. Dans ces cas, le paramètre Data Size est utilisé pour représenter le nombre d'octets requis par caractère ou entrée de codage.

[b] S'il n'est pas spécifié de Minimum Value et/ou de Maximum Value, la valeur minimale et/ou maximale pour la valeur de donnée de paramètre est/sont telle(s) que définie(s) dans la CEI 61158-5-2.

Tableau 123 – Attributs de formule de mise à l'échelle

Attribut	Signification
Multiplier Value	(Mult) Valeur de multiplicateur pour la formule de mise à l'échelle. Cette valeur peut être basée sur un autre paramètre.
Divisor Value	(Div) Valeur de diviseur pour la formule de mise à l'échelle. Cette valeur peut être basée sur un autre paramètre spécifié dans le Divisor Link.
Base Value	(Base) Valeur de base pour la formule de mise à l'échelle. Cette valeur peut être basée sur un autre paramètre spécifié dans le Base Link.
Offset Value	(Offset) Valeur de décalage pour la formule de mise à l'échelle. Cette valeur peut être basée sur un autre paramètre spécifié dans l'Offset Link. Cet attribut est un INT et peut être négatif.
Decimal Precision	(Precision) Valeur de précision dans la formule de mise à l'échelle. Cette valeur ne peut pas être basée sur un autre paramètre.

Liaisons de mise à l'échelle

Les valeurs de mise à l'échelle (multiplicateur, diviseur, base et décalage) peuvent aussi être basées sur d'autres paramètres. Les liaisons de mise à l'échelle spécifient de quelle instance la valeur de mise à l'échelle en question doit être récupérée. Si l'attribut de mise à l'échelle est mis à 0, cette valeur de mise à l'échelle est une constante et n'est liée à aucun autre paramètre. Le Tableau 124 spécifie les liaisons de mise à l'échelle.

Tableau 124 – Liaisons de mise à l'échelle

Attribut	Signification
Multiplier Link	Spécifie l'instance de paramètre d'où la valeur de multiplicateur (Multiplier Value) est récupérée.
Divisor Link	Spécifie l'instance de paramètre d'où la valeur de diviseur (Divisor Value) est récupérée.
Base Link	Spécifie l'instance de paramètre d'où la valeur de base (Base Value) est récupérée.
Offset Link	Spécifie l'instance de paramètre d'où la valeur de décalage (Offset Value) est récupérée.

Attributs de facteur d'échelle

Le facteur d'échelle représente les valeurs de paramètre UINT et INT effectives dans d'autres formats. La Formule (1) doit être utilisée pour déterminer la valeur d'étude à partir de la valeur effective.

$$\text{EngValue} = \frac{(\text{ActualValue} + \text{Offset}) \times \text{Mult} \times \text{Base}}{\text{Div}} \qquad (1)$$

La valeur d'étude peut ensuite être affichée à l'utilisateur dans les termes spécifiés dans l'attribut d'instance Decimal Precision. L'inverse de la formule de mise à l'échelle doit être utilisé pour déterminer la valeur effective à partir de la valeur d'étude (voir la Formule (2)).

$$\text{ActualValue} = \frac{(\text{EngValue} \times \text{Div})}{\text{Mult} \times \text{Base}} - \text{Offset} \qquad (2)$$

Le facteur d'échelle de précision étendue ajoute de la précision étendue au facteur d'échelle normalisé. La Formule (3) doit être utilisée pour déterminer la valeur d'étude à partir de la valeur effective.

$$\text{EngValue} = \frac{(\text{ActualValue} + \text{Offset}) \times \text{Mult} \times \text{Base}}{\text{Div} \times 10^{\text{Precision}}} \quad (3)$$

La valeur d'étude peut ensuite être affichée à l'utilisateur dans les termes spécifiés dans l'attribut Decimal Precision. L'inverse de la formule de mise à l'échelle de précision étendue doit être utilisé pour déterminer la valeur effective à partir de la valeur d'étude (voir la Formule (4)).

$$\text{ActualValue} = \frac{(\text{EngValue} \times \text{Div} \times 10^{\text{Precision}})}{\text{Mult} \times \text{Base}} - \text{Offset} \quad (4)$$

4.1.8.7.2 Services communs

Réponse Get_Attribute_All

Au **niveau de la classe**, l'ordre des attributs retournés dans le paramètre “Attribute Data” de la réponse Get_Attribute_All doit tel que défini dans le Tableau 125. Les valeurs par défaut doivent être insérées pour tous les attributs non pris en charge.

Tableau 125 – Données de réponse spécifiques à un service/objet de niveau classe pour Get_Attribute_All

Identificateur d'attribut	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Revision (octet de poids faible) Valeur par défaut = 1							
	1	Revision (octet de poids fort) Valeur par défaut = 0							
2	2	Max Instance (octet de poids faible)							
	3	Max Instance (octet de poids fort)							
8	4	Parameter Class Descriptor (octet de poids faible)							
	5	Parameter Class Descriptor (octet de poids fort)							
9	6	Configuration Assembly Instance (octet de poids faible)							
	7	Configuration Assembly Instance (octet de poids fort)							
10	8	Valeur par défaut de Native Language = 0							

Au **niveau instance**, la valeur de l'attribut Parameter Class Descriptor dicte quels attributs le service Get_Attribute_All retourne, comme suit:

- si le bit "Supports Full Attributes" (Prend en charge les attributs complets) de cet attribut n'est pas mis, le service Get_Attribute_All retourne uniquement les attributs mis en œuvre marqués Stub (Raccourci) (numéros d'attribut 1 à 6);
- si le bit "Supports Full Attributes" (Prend en charge les attributs complets) de cet attribut n'est pas mis, le service Get_Attribute_All retourne tous les attributs mis en œuvre (numéros 1 à 24).

Pour les objets Parameter Stub, l'ordre des attributs retournés dans le paramètre "Attribute Data" de la réponse Get_Attribute_All doit tel que défini dans le Tableau 126.

Tableau 126 – Données de réponse spécifiques à un service/objet de niveau instance pour Get_Attribute_All (objet Parameter Stub)

Identificateur d'attribut	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Parameter Value (premier octet)							
		...							
	n	Parameter Value (dernier octet)							
2	n+1	Link Path Size							
3	n+1	Link Path (premier octet)							
		...							
	...	Link Path (dernier octet)							
4		Descriptor (octet de poids faible)							
		Descriptor (octet de poids fort)							
5		Data Type							
6		Data Size							

Pour les objets Parameter Full, l'ordre des attributs retournés dans le paramètre "Attribute Data" de la réponse Get_Attribute_All doit tel que défini dans le Tableau 127.

Tableau 127 – Données de réponse spécifiques à un service/objet de niveau instance pour Get_Attribute_All (objet Parameter Full)

Identificateur d'attribut	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Parameter Value (premier octet)							
		...							
	n	Parameter Value (dernier octet)							
2	n+1	Link Path Size							
3	n+1	Link Path (premier octet)							
		...							
	...	Link Path (dernier octet)							
4		Descriptor (octet de poids faible)							
		Descriptor (octet de poids fort)							
5		Data Type							
6		Data Size							
7		Parameter Name String (charcount) Valeur par défaut = 0							
		Parameter Name String (premier octet)							
		...							
		Parameter Name String (dernier octet)							
8		Units String (charcount) Valeur par défaut = 0							
		Units String (premier octet)							
		...							
		Units String (dernier octet)							
9		Help String (charcount) Valeur par défaut = 0							
		Help String (premier octet)							
		...							
		Help String (dernier octet)							
10		Minimum Value (octet de poids faible) Valeur par défaut = 0							
		Minimum Value (octet de poids fort) Valeur par défaut = 0							
11		Maximum Value (octet de poids faible) Valeur par défaut = 0							
		Maximum Value (octet de poids fort) Valeur par défaut = 0							
12		Default Value (octet de poids faible) Valeur par défaut = 0							
		Default Value (octet de poids fort) Valeur par défaut = 0							
13		Scaling Multiplier (octet de poids faible) Valeur par défaut = 1							
		Scaling Multiplier (octet de poids fort) Valeur par défaut = 0							
14		Scaling Divisor (octet de poids faible) Valeur par défaut = 1							
		Scaling Divisor (octet de poids fort) Valeur par défaut = 0							
15		Scaling Base (octet de poids faible) Valeur par défaut = 1							
		Scaling Base (octet de poids fort) Valeur par défaut = 0							
16		Scaling Offset (octet de poids faible) Valeur par défaut = 0							
		Scaling Offset (octet de poids fort) Valeur par défaut = 0							
17		Multiplier Link (octet de poids faible) Valeur par défaut = 0							
		Multiplier Link (octet de poids fort) Valeur par défaut = 0							
18		Divisor Link (octet de poids faible) Valeur par défaut = 0							
		Divisor Link (octet de poids fort) Valeur par défaut = 0							

Identificateur d'attribut	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
19		Base Link (octet de poids faible) Valeur par défaut = 0							
		Base Link (octet de poids fort) Valeur par défaut = 0							
20		Offset Link (octet de poids faible) Valeur par défaut = 0							
		Offset Link (octet de poids fort) Valeur par défaut = 0							
21		Valeur par défaut de Decimal Precision = 0							
22		International Parameter Name[a]							
23		International Engineering Units[a]							
24		International Help String[a]							

[a] Dans les appareils dans lesquels les attributs International String sont mis en œuvre, les comptes de caractères de Parameter Name, Units String et Help String doivent être signalés comme étant égaux à zéro. Dans les appareils qui ne prennent pas en charge les attributs International String, chacun des attributs International String doit être signalé comme correspondant à un octet unique de zéro ou peut être absent de la réponse. Si l'un quelconque des attributs International String est pris en charge, les trois attributs doivent être signalés dans la réponse.

4.1.8.7.3 Service spécifique à un objet

Get_Enum_String

Le corps de Get_Enum_String_RequestPDU doit être tel que spécifié dans le Tableau 128.

Tableau 128 – Structure du corps de Get_Enum_String_RequestPDU

Nom	Type de données	Sémantique des valeurs
Enumerated String Number	USINT	Nombre de chaînes énumérées à récupérer. Plage de valeurs maximale possible = 0 à 255. La valeur indique le décalage de bits (par rapport à zéro) pour les énumérations de bits, la valeur effective (par rapport à zéro) pour les énumérations de valeurs.

Le corps de Get_Enum_String_ResponsePDU doit être tel que spécifié dans le Tableau 129.

Tableau 129 – Structure du corps de Get_Enum_String_ResponsePDU

Nom	Type de données	Sémantique des valeurs
Enumerated String	SHORT_STRING	Chaînes énumérées. Nombre maximal de caractères = 16

Les chaînes énumérées sont des chaînes lisibles par l'homme qui décrivent soit un bit, soit une valeur, cela dépendant du type de données du paramètre. La relation du type de chaîne au type de données de paramètre est montrée dans le Tableau 130.

Tableau 130 – Type de chaîne énumérée contre type de données de paramètre

Si le type de données du paramètre est:	les chaînes énumérées retournées sont:
OCTET, WORD, DWORD, LWORD	Chaînes énumérées de bits
USINT, SINT, UINT, INT, BOOL	Chaînes énumérées de valeurs
Tous les autres types de données	Non valide pour l'énumération

Si le type de données du paramètre est OCTET, WORD, DWORD ou LWORD, le fait de demander un service Get_Enum_String avec un numéro de chaîne énumérée de 0 sur un paramètre retourne un SHORT_STRING qui décrit le bit 0. Le fait de demander un service Get_Enum_String avec un numéro de chaîne énumérée de 1 sur un paramètre retourne un SHORT_STRING qui décrit le bit 1, et ainsi de suite jusqu'au numéro de bit maximal pris en charge par le paramètre.

Si le type de données du paramètre est SINT, USINT, INT, UINT ou BOOL, le fait de demander un service Get_Enum_String avec un numéro de chaîne énumérée de 0 sur un paramètre retourne un SHORT_STRING qui décrit la valeur égale à 0. Le fait de demander un service Get_Enum_String avec un numéro de chaîne énumérée de 1 sur un paramètre retourne un SHORT_STRING qui décrit la valeur égale à 1.

4.1.8.8 Eléments de syntaxe spécifiques à un Connection Manager

4.1.8.8.1 Attributs

4.1.8.8.1.1 Attributs de classe

Le format des attributs de classe d'objets Connection Manager doit être tel que spécifié dans le Tableau 131.

Tableau 131 – Attributs de classe d'objets Connection Manager

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
1	Revision	UINT	Révision de cet objet. Valeur = 1
2	Max Instance	UDINT	
4	Optional Attribute List	STRUCT of	
	Number attributes	UINT	
	Optional attributes	ARRAY of UINT	

4.1.8.8.1.2 Attributs d'instance

Le format des attributs d'instance d'objet Connection Manager est tel que spécifié dans le Tableau 132.

Tableau 132 – Attributs d'instance d'objet Connection Manager

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
1	OpenReqs	UINT	
2	OpenFormat Rejects	UINT	
3	OpenResource Rejects	UINT	
4	OpenOther Rejects	UINT	
5	CloseReqs	UINT	

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
6	CloseFormat Rejects	UINT	
7	CloseOther Rejects	UINT	
8	ConnTimeouts	UINT	Le nombre total de temporisations de connexion qui se sont produites dans les connexions contrôlées par ce Connection Manager.
9	Connection Entry List	STRUCT of	
	NumConnEntries	UINT	Nombre de bits utilisés dans l'élément ConnOpenBits. Cet attribut, divisé par 8 et incrémenté pour tout reste éventuel, donne la longueur en octets de la matrice dans le champ ConnOpenBits.
	ConnOpenBits	ARRAY of BOOL	Champ de bits. Liste de données de connexion. Chaque bit représente une connexion possible. 0 = Aucune connexion. 1 = Connexion établie
10	Réservé / Désuet		
11	CpuUtilization [a]	UINT	Utilisation de l'UCT en dixièmes de pourcent. Plage de 0 à 1 000 représentant 0 % à 100 %.
12	MaxBuffSize [a]	UDINT	La quantité d'espace tampon est en octets.
13	BufSize Remaining [a]	UDINT	La quantité d'espace tampon est en octets.

[a] La signification de ces valeurs et la méthode utilisée pour les calculer sont spécifiques à un vendeur.

4.1.8.8.2 Services communs

4.1.8.8.2.1 Réponse Get_Attribute_All

Au **niveau de la classe**, l'ordre des attributs retournés dans le paramètre “Attribute Data” de la réponse Get_Attribute_All doit tel que défini dans le Tableau 133.

Tableau 133 – Données de réponse spécifiques à un service/objet de niveau classe pour Get_Attribute_All

Identificateur d'attribut	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Revision (octet de poids faible) Valeur par défaut = 1							
	1	Revision (octet de poids fort) Valeur par défaut = 0							
2	2	Max Instance (octet de poids faible) Valeur par défaut = 1							
	3	Max Instance (octet de poids fort) Valeur par défaut = 0							
6	4	Max ID Number of Class Attributes (octet de poids faible) Valeur par défaut = 0							
	5	Max ID Number of Class Attributes (octet de poids fort) Valeur par défaut = 0							
7	6	Max ID Number of Instance Attributes (octet de poids faible) Valeur par défaut = 0							
	7	Max ID Number of Instance Attributes (octet de poids fort) Valeur par défaut = 0							

Au **niveau de l'instance**, l'ordre des attributs retournés dans le paramètre “Attribute Data” de la réponse Get_Attribute_All doit tel que défini dans le Tableau 134.

Tableau 134 – Données de réponse spécifiques à un service/objet de niveau instance pour Get_Attribute_All

Identificateur d'attribut	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	OpenReqs (valeur par défaut = 0)							
	1								
2	2	OpenFormat Rejects (valeur par défaut = 0)							
	3								
3	4	OpenResource Rejects (valeur par défaut = 0)							
	5								
4	6	OpenOther Rejects (valeur par défaut = 0)							
	7								
5	8	CloseReqs (valeur par défaut = 0)							
	9								
6	10	CloseFormat Rejects (valeur par défaut = 0)							
	11								
7	12	CloseOther Rejects (valeur par défaut = 0)							
	13								
8	14	ConnTimeouts (valeur par défaut = 0)							
	15								
9	16	Connection Entry List – NumConnEntries, n octets follow (valeur par défaut = 0)							
	17								
	18	Connection Entry List – ConnOpenBits (jusqu'au premier groupe de 8 BOOL, si applicable)							
	19	Connection Entry List – ConnOpenBits (jusqu'au deuxième groupe de 8 BOOL, si applicable)							
	17+n	Connection Entry List – ConnOpenBits (jusqu'au n^{e} groupe de 8 BOOL, si applicable)							
11	18+n	CpuUtilization (valeur par défaut = 0)							
	18+n+1								
12	18+n+2	MaxBuffSize (valeur par défaut = 0)							
	18+n+3								
12	18+n+4								
	18+n+5								
13	18+n+6	BufSize Remaining (valeur par défaut = 0)							
	18+n+7								
12	18+n+8								
	18+n+9								

4.1.8.8.2.2 Demande Set_Attribute_All

Au **niveau instance**, l'ordre des attributs retournés dans le paramètre "Attribute Data" de la demande Set_Attribute_All doit tel que défini dans le Tableau 135.

Tableau 135 – Données de demande spécifiques à un service/objet de niveau instance pour Set_Attribute_All

Identificateur d'attribut	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	OpenReqs[a]							
	1								
2	2	OpenFormat Rejects[a]							
	3								
3	4	OpenResource Rejects[a]							
	5								
4	6	OpenOther Rejects[a]							
	7								
5	8	CloseReqs[a]							
	9								
6	10	CloseFormat Rejects[a]							
	11								
7	12	CloseOther Rejects[a]							
	13								
8	14	ConnTimeouts[a]							
	15								

[a] Si cet attribut n'est pas pris en charge, la valeur envoyée doit être ignorée. Cette condition n'entraîne pas l'échec de la demande de service.

4.1.8.9 Eléments de syntaxe spécifiques à un objet Connection

4.1.8.9.1 Attributs

4.1.8.9.1.1 Attributs de classe

Le format des attributs de classe d'objets Connection doit être tel que spécifié dans le Tableau 136.

Tableau 136 – Attributs de classe d'objets Connection

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
1	Revision	UINT	Révision de cet objet. Valeur = 1
2	Max Instance	UINT	

4.1.8.9.1.2 Attributs d'instance

Le format des attributs d'instance d'objet Connection est tel que spécifié dans le Tableau 137.

Tableau 137 – Attributs d'instance d'objet Connection

ID d'attr	Nom	Type de données	Sémantique
1	State	USINT	Etat de l'objet.
2	Instance_type	USINT	Indique soit E/S, soit Messaging Connection (connexion de messagerie).
3	TransportClass_trigger	SWORD	Définit le comportement de la connexion

ID d'attr	Nom	Type de données	Sémantique
4	CP2/3_produced_ connection_id	UINT	Placé dans le CAN Identifier Field (champ identificateur de CAN) de l'ISO 11898 lorsque la connexion émet sur un sous-réseau CP 2/3.
5	CP2/3_consumed_ connection_id	UINT	La valeur de CAN Identifier Field de l'ISO 11898 qui désigne le message à recevoir sur un sous-réseau CP 2/3.
6	CP2/3_initial_comm_ characteristics	SWORD	Définit le(s) Message Group à travers le(s)quel(s) les productions et les consommations associées à cette connexion ont lieu sur un sous-réseau CP 2/3.
7	Produced_connection_size	UINT	Nombre maximal d'octets émis à travers cette connexion.
8	Consumed_connection_size	UINT	Nombre maximal d'octets reçus à travers cette connexion.
9	Expected_packet_rate	UINT	Définit la synchronisation associée à cette connexion.
10	CFP2_produced_connection _id	UDINT	Identifie le message envoyé sur le sous-réseau par cette connexion.
11	CPF2_consumed_connection _id	UDINT	Identifie le message reçu en provenance du sous-réseau pour cette connexion.
12	Watchdog_timeout_action	USINT	Définit comment gérer les temporisations d'inactivité/chien de garde.
13	Produced_connection_path_ length	UINT	Nombre d'octets dans l'attribut produced_connection_path.
14	Produced_connection_path	Packed EPATH	Spécifie l'/les objet(s) d'application dont les données doivent être produites par cet objet Connection. Voir 4.1.9.
15	Consumed_connection_path _ length	UINT	Nombre d'octets dans l'attribut consumed_connection_path.
16	Consumed_connection_path	Packed EPATH	Spécifie l'/les objet(s) d'application qui doivent recevoir les données consommées par cet objet Connection. Voir 4.1.9.
17	Production_inhibit_time	UINT	Définit le temps minimal entre productions de nouvelles données. Cet attribut est requis pour toutes les connexions Client E/S, excepté celles ayant un déclencheur de production "Cyclic".
18	Connection_timeout_multipli er	USINT	Spécifie le multiplicateur appliqué à la valeur expected_packet_rate pour dériver la valeur de l'Inactivity/Watchdog Timer.
19	Connection_binding_list	STRUCT UINT Array of UINT	Liste des instances de connexion E/S liées à cette instance.

State

Cet attribut définit l'état courant de l'instance de Connection. Le Tableau 138 définit les états possibles et attribue une valeur utilisée pour indiquer l'état en question.

Tableau 138 – Valeurs attribuées à l'attribut d'état

Valeur	Nom d'état	Description
00	Non-existent (Inexistant)	La connexion n'a pas encore été instanciée.
01	Configuring (En configuration)	La connexion a été instanciée et attend que les événements suivants se produisent: (1) être configurée correctement et (2) être invitée à appliquer la configuration.
02	Waiting For Connection ID (En attente de l'ID de connexion)[a]	L'instance de Connection attend exclusivement la détermination de son/ses attribut(s) consumed_connection_id et/ou produced_connection_id.[b]
03	Established (Etablie)	La connexion a été configurée de façon valide/complète et la configuration a été appliquée avec succès.
04	Timed Out (Temporisée)	Si un objet Connection subit une temporisation d'inactivité/chien de garde, une transition peut être effectuée vers cet état. Pour plus de détails, voir la description de l'attribut watchdog_timeout_action et celle de Inactivity/Watchdog.
05	Deferred Delete (Suppression différée)[a]	Si un objet Explicit Messaging Connection (connexion de messagerie explicite) subit une temporisation d'inactivité/chien de garde, une transition peut être effectuée vers cet état. Pour plus de détails, voir la description de l'attribut watchdog_timeout_action et celle de Inactivity/Watchdog Timer.
06	Closing (En cours de fermeture)	Un objet Bridged Connection (connexion pontée) a reçu, et traite, un service Forward_Close provenant du Connection Manager. La suppression de la connexion ne se produit pas tant qu'une réponse Forward_Close réussie n'a pas été reçue du nœud cible.

[a] Cette valeur est seulement utilisée sur CP 2/3.

[b] Lorsque les attributs d'instance Connection sont appliqués, il peut ne pas être possible pour le module de générer les valeurs de produced_connection_id et/ou de consumed_connection_id (voir l'attribut initial_comm_characteristics pour les règles). Si tel est le cas, toutes les tâches requises pour appliquer les attributs sont exécutées à l''exception de l'initialisation de ces attributs dans l'objet Connection et les objets Producteur/Consommateur de liaison associés et une transaction est effectuée vers cet état. Dans cet état, l'instance de Connection attend exclusivement la détermination de son/ses attribut(s) produced_connection_id et/ou consumed_connection_id.

Important: Une instance de connexion créée de manière dynamique est l'enfant de la connexion de messagerie explicite à travers laquelle elle a été créée. Des types de sous-réseau différents peuvent fournir d'autres mécanismes pour créer une connexion qui impliquent une relation parent-enfant particulière. Pour des détails, voir les spécifications spécifiques à un réseau.

Important: Toutes les ressources associées à une Connection Instance (*A)* qui a été créée de manière dynamique à travers une Explicit Messaging Connection (*B*) doivent être libérées si l'Explicit Messaging Connection (*B)* expire avant que la Connection Instance créée de manière dynamique (*A)* ne passe à l'état Established.

Important: Lorsqu'une transition est effectuée vers l'état Established, tous les temporisateurs associés à l'objet Connection sont activés (voir Connection Timing (Temporisation de connexion) dans la CEI 61158-5-2, Paragraphe 6.2.3.2.1.7).

Instance_type

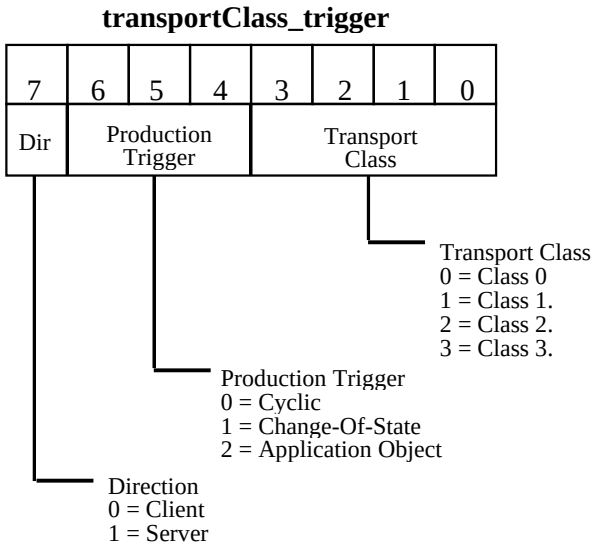
Cet attribut définit le type d'instance (voir le Tableau 139).

Tableau 139 – Valeurs attribuées à l'attribut instance_type

Valeur	Signification
00	Explicit Messaging (Messagerie explicite). Cette instance de Connection représente l'un des points d'extrémité d'une Explicit Messaging Connection. Une Explicit Messaging Connection est créée de manière dynamique en envoyant l'*Open Explicit Messaging Connection Request* (demande d'ouvrir une connexion de messagerie explicite) à la Connection Class (classe de connexion).
01	I/O (E/S). Cette instance de Connection représente l'un des points d'extrémité d'une I/O Connection (Connexion E/S). Une I/O Connection est créée de manière dynamique en envoyant une *Create Request* à la Connection Class.
02	Bridged (Pontée). Cette instance de Connection représente un saut intermédiaire d'une connexion I/O ou Explicit Messaging pontée. Une paire d'objets Bridged Connection (un pour chaque sous-réseau ponté entre eux) est créée de manière dynamique par un nœud après avoir reçu avec succès un service Forward_Open vers l'objet Connection Manager lorsque ce nœud n'est pas le point d'extrémité (la cible).

TransportClass_trigger

Définit s'il s'agit d'une connexion productrice seulement, consommatrice seulement, ou à la fois productrice et consommatrice. Si ce point d'extrémité doit effectuer une production de données, cet attribut définit aussi l'événement qui déclenche la production. Les huit (8) bits sont divisés comme montré à la Figure 8.



Légende

Anglais	Français
Dir	Sens
Production Trigger	Déclencheur de production
Transport Class	Classe de transport
Class	Classe
Cyclic	Cyclique
Change-Of-State	Changement d'état
Application Object	Objet d'application
Direction	Sens
Client	Client
Server	Serveur

Figure 8 – Attribut Transport Class Trigger

Le **bit Direction** de l'octet transportClass_trigger indique si le point d'extrémité doit agir comme Client ou comme Serveur sur cette connexion (voir le Tableau 140).

Tableau 140 – Valeurs possibles dans le bit Direction

Valeur	Signification	
0	Client	Ce point d'extrémité donne le comportement en Client associé à cette connexion. En outre, cette valeur indique que les bits *Production Trigger* dans l'octet transportClass_trigger contiennent la description de l'instant où le Client doit produire le message associé à cette connexion. Les connexions clientes avec la valeur du déclencheur de production de 0 ou 1 (Cyclic ou Change-of-State) doivent produire immédiatement après la transition vers l'état Established.
1	Serveur	Ce point d'extrémité donne le comportement en Serveur associé à cette connexion. En outre, cette valeur indique que les bits *Production Trigger* dans l'octet transportClass_trigger doivent être IGNORED (ignorés). Les bits *Production Trigger* sont ignorés en raison du fait qu'un point d'extrémité Serveur *réagit* à l'émission à partir du Client. Le seul moyen par lequel un point d'extrémité Serveur est déclenché pour émettre est lorsque cette *réaction* appelle la production d'un message (classes de transport 2 ou 3).

Le Tableau 141 énumère les valeurs qui sont possibles dans les bits **Production Trigger** de l'attribut **transportClass_trigger**.

Tableau 141 – Valeurs possibles dans les bits Production Trigger

Si la valeur est:	La production d'un message est:	
0	Cyclic	L'expiration du temporisateur Transmission Trigger Timer déclenche la production de données. Pour une description détaillée du Transmission Trigger Timer, voir la CEI 61158-5-2, Paragraphe 6.2.3.2.1.7 Connection Timing (Temporisation de connexion).
1	ChangeOf-State	La production a lieu lorsqu'un changement d'état est détecté par l'objet d'application. NOTE Le point d'extrémité consommateur peut avoir été configuré pour attendre le paquet à un certain débit, quel que soit le mécanisme de déclenchement au point d'extrémité producteur. Pour plus d'informations, voir la description de l'attribut expected_packet_rate d'un objet Connection et la description de Connection Timing dans la CEI 61158-5-2, Paragraphe 6.2.3.2.1.7.
2	Application Object Triggered	L'objet d'application décide du moment auquel déclencher la production. NOTE Le point d'extrémité consommateur peut avoir été configuré pour attendre le paquet à un certain débit, quel que soit le mécanisme de déclenchement au point d'extrémité producteur. Pour plus d'informations, voir la description de l'attribut expected_packet_rate d'un objet Connection et la description de Connection Timing dans la CEI 61158-5-2, Paragraphe 6.2.3.2.1.7.
3 - 7	Réservé	

Le Tableau 142 énumère les valeurs possibles dans le quartet **Transport Class** de l'attribut **transportClass_trigger**. Les comportements résultant de ces valeurs particulières sont illustrés dans la série de figures qui suivent le tableau.

Tableau 142 – Valeurs possibles dans les bits Transport Class

Valeur	Signification	
0	Classe de transport 0	En fonction de la valeur du bit *Dir*, ce point d'extrémité de connexion sera un point d'extrémité producteur seulement OU consommateur seulement. A la suite de l'application de cette instance de Connection, le module instancie soit un Link Producer (bit *Dir* = Client, producteur seulement), soit un Link Consumer (bit *Dir* = Server, consommateur seulement) devant être associé à cette connexion.
1	Classe de transport 1	
2	Classe de transport 2	Indique que le module produira ET consommera à travers cette connexion. Le point d'extrémité Client génère la première production de données qui est consommée par le Server, amenant le Server à retourner une production qui est consommée par le Client.
3	Classe de transport 3	
4	Classe de transport 4	Non bloquant.
5	Classe de transport 5	Non bloquant, fragmentant.
6	Classe de transport 6	Multidiffusion, fragmentant.
7 à F	Réservé	

Une valeur de compte de séquences de 16 bits est ajoutée en préfixe au début de tous les transports de classes 1, 2 et 3. Cette valeur est utilisée pour détecter la livraison de doublons de paquets de données. Les valeurs du compte de séquence sont initialisées lors de la production du premier message et déterminées par le type de transport et de déclencheur quand les messages ultérieurs sont produits. Un renvoi d'anciennes données pour conserver la connexion ne doit pas changer le compte de séquences et un consommateur doit ignorer les données lorsqu'elles sont reçues avec un doublon de compte de séquences. Les applications consommatrices peuvent utiliser ce mécanisme pour distinguer entre échantillons nouveaux et échantillons anciens qui avaient été envoyés pour maintenir la connexion.

Cependant, sur CP 2/3, les classes de transport 2 et 3 n'ajoutent pas en préfixe un compte de séquences de 16 bits comme décrit en 4.1.4.

Les tableaux et figures suivants illustrent les combinaisons valides de **Production Trigger** et de **Transport Class**, et donnent une description des comportements de Client et de Server. Voir la CEI 61158-5-2, Paragraphe 6.2.3.2.1.7, pour une description du **Transmission Trigger Timer**, qui est montré dans les illustrations.

Le Tableau 143 résume les valeurs valides de l'attribut **transportClass_trigger** d'une Connection Instance:

Tableau 143 – Valeurs de l'attribut TransportClass_Trigger

Bits TransportClass_trigger	Signification
1 xxx 0000	Direction = Server, Production Trigger = IGNORED, Transport Class = 0.
1 xxx 0001	Direction = Server, Production Trigger = IGNORED, Transport Class = 1.
1 xxx 0010	Direction = Server, Production Trigger = IGNORED, Transport Class = 2.
1 xxx 0011	Direction = Server, Production Trigger = IGNORED, Transport Class = 3. Il s'agit de la valeur attribuée à cet attribut au sein du point d'extrémité Server d'une Explicit Messaging Connection.
0 000 0000	Direction = Client, Production Trigger = Cyclic, Transport Class = 0.
0 000 0001	Direction = Client, Production Trigger = Cyclic, Transport Class = 1.
0 000 0010	Direction = Client, Production Trigger = Cyclic, Transport Class = 2.
0 000 0011	Direction = Client, Production Trigger = Cyclic, Transport Class = 3.

Bits TransportClass_trigger	Signification
0 001 0000	Direction = Client, Production Trigger = ChangeOfState, Transport Class = 0.
0 001 0001	Direction = Client, Production Trigger = ChangeOfState, Transport Class = 1.
0 001 0010	Direction = Client, Production Trigger = ChangeOfState, Transport Class = 2.
0 001 0011	Direction = Client, Production Trigger = ChangeOfState, Transport Class = 3.
0 010 0000	Direction = Client, Production Trigger = Application object, Transport Class = 0.
0 010 00001	Direction = Client, Production Trigger = Application object, Transport Class = 1
0 010 0010	Direction = Client, Production Trigger = Application object, Transport Class = 2.
0 010 0011	Direction = Client, Production Trigger = Application object, Transport Class = 3. Il s'agit de la valeur attribuée à cet attribut au sein du point d'extrémité Client d'une Explicit Messaging Connection.
1 111 1111	Valeur par défaut attribuée à cet attribut dans une connexion E/S.

Le Tableau 144 résume le comportement du client pour la Classe 0 pour ce qui est de la production de message pour les différents types de déclencheurs.

Tableau 144 – Résumé du comportement du client, Classe de transport 0

Type de déclencheur	Production déclenchée par	Production répétée pour conserver la connexion
Cyclic	Temporisateur d'émission (API)	n/a
Change of State	Changement détecté de données	Temporisateur d'émission (API)
Application	Mise à jour des données de l'application	Temporisateur d'émission (API)

Le Tableau 145 résume le comportement du client pour les Classes 1, 2 et 3 pour ce qui est de la production de message et du décompte des séquences pour les différents types de déclencheurs.

Tableau 145 – Résumé du comportement du client, Classe de transport 1, 2 et 3

Type de déclencheur	Compte de séquence incrémenté de	Production déclenchée par	Production répétée pour conserver la connexion
Cyclic	Mise à jour des données de l'application	Temporisateur d'émission (API)	n/a
Change of State	Changement détecté de données	Changement détecté de données	Temporisateur d'émission (API)
Application	Mise à jour des données de l'application	Mise à jour des données de l'application	Temporisateur d'émission (API)

CP2/3_produced_connection_id

Cet attribut est Requis pour un sous-réseau CP 2/3. Les autres types de sous-réseau ne doivent pas utiliser cet attribut.

Il contient l'ID de connexion CP 2/3 à associer aux émissions envoyées sur cette connexion (le cas échéant), à savoir, la valeur qui sera spécifiée dans le CAN Identifier Field de l'ISO 11898 lorsque cette Connection émet. Voir la CEI 62026-3:2008, 5.1.2 pour une description de la manière dont CP 2/3 utilise le champ CAN Identifier Field. Cette valeur est

chargée directement dans l'attribut associé connection_id du Link Producer. Les valeurs sont définies dans le Tableau 146.

Tableau 146 – Valeurs définies pour l'attribut CP2/3_produced_connection_id

Valeur	Signification
0 à $7F0_{hex}$	La valeur à placer dans le CAN Identifier Field lorsque cette connexion émet.
800_{hex} à $FFFE_{hex}$	Réservé
$FFFF_{hex}$	Valeur par défaut attribuée à cet attribut dans une connexion E/S. Cet attribut retiendra cette valeur si cette instance de Connection ne produit pas de donnée (consommateur seulement).

CP2/3_consumed_connection_id

Cet attribut est Requis pour un sous-réseau CP 2/3. Les autres types de sous-réseau ne doivent pas utiliser cet attribut.

Il contient l'ID de connexion CP qui identifie les messages devant être reçus sur cette connexion (le cas échéant), à savoir, la valeur de CAN Identifier Field qui est associée aux messages que cet objet Connection reçoit. Voir la CEI 62026-3:2008, 5.1.2 pour une description de la manière dont CP 2/3 utilise le champ CAN Identifier Field. Cette valeur est chargée directement dans l'attribut associé connection_id du Link Consumer. Les valeurs sont définies dans le Tableau 147.

Tableau 147 – Valeurs définies pour l'attribut CP2/3_consumed_connection_id

Valeur	Signification
0 à $7F0_{hex}$	La valeur qui identifie les messages à consommer. Elle sera spécifiée dans le CAN Identifier Field des messages qui sont à consommer.
800_{hex} à $FFFE_{hex}$	Réservé
$FFFF_{hex}$	Valeur par défaut attribuée à cet attribut dans une connexion E/S. Cet attribut retiendra cette valeur si cette instance de Connection ne consomme pas de donnée (producteur seulement).

CP2/3_initial_comm_characteristics

Cet attribut est Requis pour un sous-réseau CP 2/3. Les autres types de sous-réseau ne doivent pas utiliser cet attribut.

Il définit le(s) Message Group à travers le(s)quel(s) les productions et les consommations associées à cette connexion ont lieu.

Cet octet est divisé en deux quartets, comme montré à la Figure 9.

7	6	5	4	3	2	1	0
Caractéristiques de production initiales				Caractéristiques de consommation initiales			

Figure 9 – Format de l'attribut CP2/3_initial_comm_characteristics

Le Tableau 148 énumère les valeurs qui sont possibles au sein du quartet (quartet supérieur) Initial Production Characteristics (Caractéristiques de production initiales) de l'attribut CP2/3_initial_comm_characteristics.

Tableau 148 – Valeurs pour le quartet Initial Production Characteristics

Valeur	Signification	
0	Produire à travers le Message Group 1	La production associée à cette connexion a lieu à travers le Message Group 1. Le module producteur génère la valeur d'ID de connexion et la charge dans l'attribut CP2/3_produced_connection_id de l'objet Connection. Le module producteur alloue un ID de message issu de son groupement d'ID de message de Groupe 1 et le combine avec son ID de MAC Source pour générer l'ID de connexion. L'ID de message de Groupe 1 ayant la valeur numérique la plus faible est à utiliser pour générer la valeur de l'attribut CP2/3_produced_connection_id. Cette valeur doit aussi être chargée dans l'/les attribut(s) CP2/3_consumed_connection_id associé(s) à l'objet ou aux objets Connection consommateur(s).
1	Produire à travers le Message Group 2 (Destination)	La production associée à cette connexion a lieu à travers le Message Group 2. Le MAC ID du destinataire prévu (Destination MAC ID) doit en outre être placé dans le composant MAC ID de Group 2 Identifier Field. Dans ce cas, le module consommateur génère la valeur de l'ID de connexion à associer aux émissions sur cette connexion. Lorsque le module consommateur a généré cette valeur et l'a chargée dans l'attribut approprié CP2/3_consumed_connection_id de l'objet Connection, elle peut être lue et ensuite chargée dans l'attribut CP2/3_produced_connection_id de l'objet Connection producteur.
2	Produire à travers le Message Group 2 (Source)	La production associée à cette connexion a lieu à travers le Message Group 2. En outre, l'ID de MAC (Source MAC ID) du module producteur est à placer au sein du composant ID de MAC de l'identificateur de groupe 2. Dans ce cas, le module producteur génère la valeur d'ID de connexion et la charge dans l'attribut CP2/3_produced_connection_id de l'objet Connection. L'ID de message de Groupe 2 ayant la valeur numérique la plus faible est à utiliser pour générer la valeur de l'attribut CP2/3_produced_connection_id. Cette valeur doit aussi être chargée dans l'/les attribut(s) CP2/3_consumed_connection_id associé(s) à l'objet ou aux objets Connection consommateur(s).
3	Produire à travers le Message Group 3	La production associée à cette connexion a lieu à travers le Message Group 3. Le module producteur génère la valeur d'ID de connexion et la charge dans l'attribut CP2/3_produced_connection_id de l'objet Connection. Le module producteur alloue un ID de message issu de son groupement d'ID de message de Groupe 3 et le combine avec son ID de MAC Source pour générer l'ID de connexion. L'ID de message de Groupe 3 ayant la valeur numérique la plus faible est à utiliser pour générer la valeur de l'attribut CP2/3_produced_connection_id. Cette valeur doit aussi être chargée dans l'/les attribut(s) CP2/3_consumed_connection_id associé(s) à l'objet ou aux objets Connection consommateur(s).
4 à 0xE	Réservé	
0xF	Valeur par défaut	La valeur par défaut attribuée au quartet Initial Production Characteristics (Caractéristiques de production initiales) de CP2/3 au sein de la connexion E/S. NOTE S'il s'agit d'une connexion E/S seulement consommatrice, la valeur par défaut reste dans ce quartet. Les objets Explicit Messaging Connection configurent automatiquement cet attribut lorsque la connexion est établie.

Le Tableau 149 énumère les valeurs possibles au sein du quartet (quartet inférieur) Initial Consumption Characteristics (Caractéristiques de consommation initiales) de l'attribut CP2/3_initial_comm_characteristics.

Tableau 149 – Valeurs pour le quartet Initial Consumption Characteristics

Valeur	Signification	
0	Consommer un message de groupe 1	Le message à consommer sera émis à travers le Message Group 1. Le module producteur génère la valeur d'ID de connexion. Cette valeur doit être chargée dans l'attribut CP2/3_consumed_connection_id associé à l'objet ou aux objets Connection consommateur(s).
1	Consommer un message de groupe 2 (Destination)	Le message à consommer sera émis à travers le Message Group 2. L'ID de MAC (Destination MAC ID) du destinataire prévu est spécifié au sein de l'identificateur de groupe 2. Le module consommateur génère la valeur d'ID de connexion et la charge dans l'attribut CP2/3_consumed_connection_id associé à cet objet Connection. L'ID de message de Groupe 2 ayant la valeur numérique la plus faible est à utiliser pour générer la valeur de l'attribut CP2/3_consumed_connection_id. Cette valeur doit être chargée dans l'attribut CP2/3_produced_connection_id de l'objet Connection producteur.
2	Consommer un message de groupe 2 (Source)	Le message à consommer sera émis à travers le Message Group 2. L'ID de MAC (Source MAC ID) du module émetteur est spécifié au sein de l'identificateur de groupe 2. Dans ce cas, le module producteur génère la valeur d'ID de connexion et la charge dans l'attribut CP2/3_produced_connection_id de l'objet Connection. Cette valeur doit être chargée dans l'attribut CP2/3_consumed_connection_id associé à l'objet ou aux objets Connection consommateur(s).
3	Consommer un message de groupe 3	Le message à consommer sera émis comme un message de groupe 3. Le module producteur génère la valeur d'ID de connexion. La valeur d'ID de connexion doit être chargée cet attribut CP2/3_consumed_connection_id de l'objet Connection.
4 à 0xE	Réservé	
0xF	Valeur par défaut	La valeur par défaut attribuée au quartet Initial Consumption Characteristics (Caractéristiques de consommation initiales) de CP 2/3 au sein de la connexion E/S. NOTE S'il s'agit d'une connexion E/S seulement productrice, la valeur par défaut reste dans ce quartet. Les Explicit Messaging Connection configurent automatiquement cet attribut lorsque la connexion est établie.

Important: Le module qui génère un ID de connexion doit garantir qu'il n'alloue pas la paire ID de message/ID de MAC de sorte que deux modules distincts soient capables d'émettre des profils binaires identiques au sein de l'Identifier Field.

Produced_connection_size

La signification de cet attribut est différente pour les Explicit Messaging Connection par rapport à celle qu'elle est pour les I/O Connection. Si le sous-réseau définit un protocole de fragmentation et l'appareil prend en charge la fragmentation, cette taille peut être supérieure à la plus grande taille de trame. Pour plus de détails, voir les spécifications d'adaptation spécifiques à un réseau.

Pour les Explicit Messaging Connections:

Cet attribut signifie le nombre maximal d'octets de données de demande/réponse de routeur de messages (Message Router Request/Response Data) (voir 4.1.8) qu'un dispositif est capable d'émettre sur cette connexion. Les appareils qui imposent une limite connue à la quantité maximale de données de demande/réponse de routeur de message qui peuvent être émises dans un seul message, ou une seule série fragmentée, initialisent cet attribut en conséquence. Les appareils qui ne peuvent pas ou ne prédéfinissent pas une limite d'émission préalable placent la valeur 0xffff dans cet attribut (il peut encore exister une limite, mais elle n'est pas connue à l'avance).

Important: En raison de la nature de la messagerie explicite (Explicit Messaging), la longueur des messages explicites variera pendant la durée de vie d'une connexion.

Les Explicit Messaging Connection accomplissent la fragmentation en fonction de la longueur du message courant à émettre lorsque le type de sous-réseau peut définir un protocole de fragmentation.

Pour les I/O Connection:

Si le **transportClass_trigger** indique que cette instance de Connection doit produire, cet attribut définit la quantité maximale de données E/S qui peuvent être produites comme un seul bloc sur cette connexion. La quantité de données E/S à transmettre à un instant donné quelconque peut être inférieure ou égale à l'attribut **connection_size**.

Cet attribut a zéro (0) comme valeur par défaut au sein de la connexion E/S. Si le type de sous-réseau prend en charge la fragmentation et cet attribut est mis à une valeur supérieure à la plus grande charge utile dans une I/O Connection, la connexion morcellera les données en plusieurs fragments. Pour plus de détails, voir les spécifications d'adaptation spécifiques à un réseau.

Important: La fragmentation au sein des connexions E/S est accomplie en fonction de la valeur dans cet attribut, quelle que soit la quantité courante de données à émettre.

Important: Les messages E/S qui ne contiennent pas de données E/S d'application et étaient configurés pour contenir des données (par le fait que produced_connection_size est supérieur à zéro (0)) sont définis pour indiquer un événement **No Data** (Aucune donnée) pour l'objet ou les objets d'application de réception. Le comportement d'un objet d'application à la détection d'un événement **No Data** est spécifique à chaque objet d'application.

Consumed_connection_size

La signification de cet attribut est différente pour les Explicit Messaging Connection par rapport à celle qu'elle est pour les I/O Connection. Si le sous-réseau définit un protocole de fragmentation et l'appareil prend en charge la fragmentation, cette taille peut être supérieure à la plus grande taille de trame. Pour plus de détails, voir les spécifications d'adaptation spécifiques à un réseau.

Pour les Explicit Messaging Connections:

Cet attribut signifie le nombre maximal d'octets de données de demande/réponse de routeur de message (Message Router Request/Response Data) qu'un appareil est capable de recevoir sur cette connexion.

Les appareils qui imposent une limite connue à la quantité maximale de données de demande/réponse de routeur de message qui peuvent être reçues dans un seul message, ou une seule série fragmentée, initialisent cet attribut en conséquence. Les appareils qui ne peuvent pas ou ne prédéfinissent pas une limite de réception préalable placent la valeur 0xffff dans cet attribut (il peut encore exister une limite, mais elle n'est pas connue à l'avance). En raison de la nature de la messagerie explicite (Explicit Messaging), la longueur des messages explicites variera pendant la durée de vie d'une connexion.

Pour les I/O Connection:

Si l'attribut **transportClass_trigger** indique que cette connexion doit consommer, cet attribut définit la quantité maximale de données qui peuvent être reçues comme un seul bloc sur cette connexion. La quantité réelle de données E/S reçues à un instant donné quelconque peut être inférieure ou égale à l'attribut **connection_size**.

Cet attribut a zéro (0) comme valeur par défaut au sein de la connexion E/S. Si le type de sous-réseau prend en charge la fragmentation et cet attribut est mis à une valeur supérieure à la plus grande charge utile **dans une I/O Connection**, la connexion traitera le protocole de fragmentation. Pour plus de détails, voir les spécifications d'adaptation spécifiques à un réseau.

La longueur d'un message E/S doit être inférieure ou égale à son attribut pour qu'un objet I/O Connection le reçoive comme étant un message valide. Si un objet I/O Connection reçoit un message dont la longueur est supérieure à cet attribut, il rejette immédiatement le message et interrompt tout traitement ultérieur.

NOTE En ce qui concerne le point d'extrémité Server d'une connexion de classe de transport 2 ou 3, cette condition d'erreur (trop de données) fait qu'aucune réponse n'est transmise et que le temporisateur chien de garde n'est pas éconduit (kicked).

Expected_packet_rate

Cet attribut est utilisé pour générer les valeurs chargées dans le **Transmission Trigger Timer** et l'**Inactivity/Watchdog Timer**. Pour une description détaillée des temporisateurs Transmission Trigger et Inactivity/Watchdog, voir la CEI 61158-5-2, Paragraphe 6.2.3.2.1.7.

La résolution de cet attribut est en millisecondes. Une demande de configurer cet attribut peut se traduire en la spécification d'une valeur temporelle à laquelle un produit ne peut pas satisfaire. En plus de l'accomplissement d'une vérification de plage spécifique à un produit lorsqu'une demande de modifier cet attribut est reçue, les étapes suivantes sont accomplies:

- Si la valeur spécifiée n'est pas égale à un incrément de la résolution d'horloge disponible, la valeur est arrondie par excès à la valeur pratique suivante. Par exemple: une demande Set_Attribute_Single reçue spécifie la valeur 5 pour l'attribut **expected_packet_rate** et le produit assure une résolution de 10 millisecondes sur les temporisateurs. Dans ce cas, le produit chargerait la valeur 10 dans l'attribut **expected_packet_rate**.
- La valeur qui est effectivement chargée dans l'attribut **expected_packet_rate** est rapportée dans le Service Data Field (champ de données de service) d'un message de réponse Set_Attribute_Single associé à une demande de modifier cet attribut.
- Si la valeur demandée est égale à un incrément de la résolution d'horloge, la valeur demandée est chargée dans l'attribut expected_packet_rate et rapportée dans la réponse. Par exemple: si la valeur 100 est demandée et la résolution d'horloge est 10 millisecondes, une valeur de 100 est chargée.
- Lorsqu'un objet Connection est dans l'état **Established**, toutes les éventuelles modifications de l'attribut **expected_packet_rate** ont un effet immédiat sur le temporisateur Inactivity/Watchdog Timer. Les étapes suivantes sont accomplies par un objet Connection dans l'état **Established** lorsqu'une demande de modification de l'attribut **expected_packet_rate** est reçue:
  - le temporisateur courant Inactivity/Watchdog Timer est annulé,
  - un nouvel Inactivity/Watchdog Timer est activé en fonction de la nouvelle valeur contenue dans l'attribut expected_packet_rate.

Cet attribut a 2500 (2 500 ms) comme valeur par défaut au sein des Explicit Messaging Connections et zéro (0) comme valeur par défaut au sein d'une I/O Connection.

CPF2_produced_connection_id

Contient l'ID de connexion, qui identifie les messages à envoyer sur cette connexion (le cas échéant). Cet attribut ne doit pas être mis en œuvre lorsque le type de sous-réseau est CP 2/3.

CPF2_consumed_connection_id

Contient l'ID de connexion, qui identifie les messages à recevoir sur cette connexion (le cas échéant). Cet attribut ne doit pas être mis en œuvre lorsque le type de sous-réseau est CP 2/3.

Watchdog_timeout_action

Cet attribut définit l'action qu'il convient que l'objet Connection accomplisse lorsque l'Inactivity/Watchdog Timer expire. Le Tableau 150 définit les spécificités de cet attribut.

Tableau 150 – Valeurs de watchdog_timeout_action

Valeur	Signification
0	*Passer à Timed Out (Temporisé)*. La connexion passe à l'état Timed Out (temporisé) et reste dans cet état jusqu'à ce qu'elle soit Reset (réinitialisée) ou Deleted (supprimée). La commande de Reset ou Delete pourrait provenir d'une source interne (par exemple, un objet application) ou du réseau (par exemple, un outil de configuration). Il s'agit de la valeur par défaut de cet attribut en ce qui concerne les I/O Connection. Cette valeur est non valide pour les Explicit Messaging Connection.
1	*Auto Delete (Suppression automatique)*. La classe de connexion supprime automatiquement la connexion si elle subit une temporisation d'inactivité/chien de garde (Inactivity/Watchdog timeout). Il s'agit de la valeur par défaut de cet attribut en ce qui concerne les Explicit Messaging Connection.
2	*Auto Reset (Réinitialisation automatique)*. La connexion reste dans l'état Established et redémarre immédiatement le temporisateur Inactivity/Watchdog timer. Cette valeur est non valide pour les Explicit Messaging Connection.
3	*Deferred Delete (Suppression différée)*. La connexion passe à l'état Deferred (différé) si des instances de connexion enfant quelconques sont dans l'état Established. Si aucune instance de connexion enfant n'est dans l'état Established, la connexion est supprimée. Cette valeur est seulement utilisée sur CP 2/3 et est non valide pour les I/O Messaging Connection.
4 à FF	Réservé

Produced_connection_path_length

Spécifie le nombre d'octets d'information au sein de l'attribut **produced_connection_path**. Il est automatiquement initialisé lorsque l'attribut **produced_connection_path** est configuré. Cet attribut a zéro (0) comme valeur par défaut.

Produced_connection_path

L'attribut **produced_connection_path** est constitué d'un train d'octets qui définit l'/les objet(s) d'application dont les données sont à produire par cet objet Connection. Le format de ce train d'octets est spécifié en 4.1.9. Comme valeur par défaut, cet attribut est vide à la suite de l'instanciation de la connexion. Il reste vide au sein des Explicit Messaging Connection et au sein des objets Connection qui ne produisent pas.

Consumed_connection_path_length

Spécifie le nombre d'octets d'information au sein de l'attribut **consumed_connection_path**. Il est automatiquement initialisé lorsque l'attribut **consumed_connection_path** est configuré. Cet attribut a zéro (0) comme valeur par défaut.

Consumed_connection_path

L'attribut **consumed_connection_path** est constitué d'un train d'octets qui définit l'/les objet(s) d'application qui doivent recevoir les données consommées par cet objet Connection. Le format de ce train d'octets est spécifié en 4.1.9. Comme valeur par défaut, cet attribut est vide à la suite de l'instanciation de la connexion. Il reste vide au sein des Explicit Messaging Connection et au sein des objets Connection qui ne consomment pas.

Production_inhibit_time

Cet attribut est utilisé pour configurer le retard minimal entre nouvelles productions de données. Il est requis pour toutes les connexions Client E/S, excepté celles ayant un déclencheur de production "Cyclic". Le service Set_Attribute_Single doit être pris en charge lorsque cet attribut est mis en œuvre. Une valeur de zéro (la valeur par défaut pour cet attribut) indique l'absence de temps de neutralisation.

La résolution de cet attribut est en millisecondes. Une demande de configurer cet attribut peut se traduire en la spécification d'une valeur temporelle à laquelle un produit ne peut pas satisfaire. En plus de l'accomplissement d'une vérification de plage spécifique à un produit lorsqu'une demande de modifier cet attribut est reçue, les étapes suivantes sont accomplies:

- Si la valeur spécifiée n'est pas égale à un incrément de la résolution d'horloge disponible, la valeur est arrondie par excès à la valeur pratique suivante. Par exemple: une demande Set_Attribute_Single reçue spécifie la valeur 5 pour l'attribut **production_inhibit_time** et le produit assure une résolution de 10 millisecondes sur les temporisateurs. Dans ce cas, le produit chargerait la valeur 10 dans l'attribut **production_inhibit_time**.
- La valeur qui est effectivement chargée dans l'attribut **production_inhibit_time** est rapportée dans le Service Data Field (champ de données de service) d'un message de réponse Set_Attribute_Single associé à une demande de modifier cet attribut.
- Si la valeur demandée est égale à un incrément de la résolution d'horloge, la valeur demandée est chargée dans l'attribut **production_inhibit_time** et rapportée dans la réponse. Par exemple: la valeur 100 est demandée et la résolution d'horloge est 10 millisecondes.

La valeur de **production_inhibit_time** est chargée dans le temporisateur Production Inhibit Timer chaque fois qu'une nouvelle production de données a lieu.

Lorsqu'un objet Connection est dans l'état **Established**, toutes les éventuelles modifications de l'attribut **production_inhibit_time** n'ont aucun effet sur le Production Inhibit Timer en fonctionnement. La nouvelle valeur de production_inhibit_time est chargée dans le temporisateur Production Inhibit Timer sur la nouvelle production suivante de données.

Lorsque le service apply_attributes est reçu, le **production_inhibit_time** doit être vérifié par rapport à l'attribut **expected_packet_rate**. Si la valeur de **expected_packet_rate** est supérieure à zéro, mais inférieure à la valeur de **production_inhibit_time**, une erreur doit être retournée. Dans ce cas, lorsque deux valeurs d'attribut sont en conflit, utiliser l'ID de l'attribut **production_inhibit_time** comme étant le code d'erreur supplémentaire envoyé dans la réponse d'erreur.

Connection_timeout_multiplier

Le Connection_timeout_multiplier spécifie le multiplicateur appliqué au débit de paquet prévu pour obtenir la valeur utilisée par le temporisateur Inactivity/Watchdog Timer. Pour les valeurs énumérées de cet attribut, voir la description du paramètre Connection Timeout Multiplier au sein de Paramètres de service spécifiques à un objet Connection Manager. La valeur par défaut pour cet attribut est zéro (spécifiant un multiplicateur de 4).

Connection_binding_list

L'attribut Connection_binding_list identifie les instances de connexion qui sont liées à cette connexion. La structure de l'attribut fournit un compte 16 bits de connexions liées, suivi d'une liste d'instances liées. Cet attribut doit être pris en charge si le service Connection_Bind est pris en charge.

4.1.8.9.2 Services spécifiques à un objet

4.1.8.9.2.1 Service Connection_Bind

Le service spécifique à un objet Connection_Bind lie ensemble deux instances de I/O Connection créées de manière dynamique à des fins de temporisations et de suppressions de connexion. Une I/O connection créée de manière dynamique est le résultat d'un service Create à la classe Connection avec l'attribut Instance_type mis à I/O (valeur d'attribut de 1). Si l'une de ces instances de I/O temporise ou est supprimée, l'autre doit manifester le même comportement.

Le corps de Connection_Bind_RequestPDU doit être tel que spécifié dans le Tableau 151.

Tableau 151 – Structure du corps de Connection_Bind_RequestPDU

Nom de paramètre	Type de données	Description de paramètre
Bound Instances	STRUCT of UINT UINT	Nombres d'instances des objets Connection à lier.

Le corps de Connection_Bind_ResponsePDU est vide.

La réponse de service peut retourner un statut issu de la liste de codes de statut dans le Tableau 152.

Tableau 152 – Statut spécifique à un objet pour le service Connection_Bind

Code de statut général	Code de statut étendu	Description de statut
0x02	0x01	L'une ou les deux instances de connexion sont inexistantes.
0x02	0x02	La classe et/ou l'instance de connexion est/sont à court de ressources pour lier des instances.
0x0C	0x01	Toutes les deux instances de connexion existent, mais l'une au moins n'est pas dans l'état Established.
0x20	0x01	Toutes les deux instances de connexion sont la même valeur.
0xD0	0x01	L'une ou les deux instances de connexion n'est pas/ne sont pas une connexion E/S créée de manière dynamique.
0xD0	0x02	L'une ou les deux instances de connexion a/ont été créée(s) en interne et l'appareil n'autorise pas une liaison à lui.

4.1.8.9.2.2 Service Producing_Application_Lookup

Le service spécifique à un objet Producing_Application_Lookup fournit un mécanisme pour trouver une ou plusieurs instances de connexion dans l'état Established produisant des données à partir d'un objet d'application donné. Si le chemin d'application productrice requis est en train d'être produit par une connexion quelconque dans l'état Established, le nombre d'instances de chaque connexion produisant à partir de ce chemin est retourné.

Le corps de Producing_Application_Lookup_RequestPDU doit être tel que spécifié dans le Tableau 153.

Tableau 153 – Structure du corps de Producing_Application_Lookup_RequestPDU

Nom de paramètre	Type de données	Description de paramètre
Producing Application Path	Packed EPATH	Le chemin de connexion de l'application productrice à rechercher au sein des connexions dans l'état Established. Le chemin doit être un simple Logical EPATH ou Symbolic path.

Le corps de Producing_Application_Lookup_ResponsePDU doit être tel que spécifié dans le Tableau 154.

Tableau 154 – Structure du corps de Producing_Application_Lookup_ResponsePDU

Nom de paramètre	Type de données	Description de paramètre
Instance Count	UINT	Nombre d'instances retourné dans le paramètre Connection Instance List.
Connection Instance List	Struct of UINT	Liste de nombres d'instances de connexion produisant les données à partir du chemin de connexion demandé au sein du nœud.

La réponse de service peut retourner un statut issu de la liste de codes de statut dans le Tableau 155.

Tableau 155 – Codes de statut du service Producing_Application_Lookup

Code de statut général	Code de statut étendu	Description de statut
0x02	0x01	Le chemin de connexion n'a été trouvé dans aucune instance dans l'état Established.

4.1.8.9.2.3 Service SafetyOpen

Voir la CEI 61784-3-2 pour la description de ce service.

4.1.8.9.2.4 Service SafetyClose

Voir la CEI 61784-3-2 pour la description de ce service.

4.1.9 Chemins de message et de connexion

4.1.9.1 Contenu

Le chemin doit spécifier l'élément d'objet qui est soit la cible d'une demande de connexion, soit la destination d'une demande de message. Le chemin doit contenir plusieurs segments qui peuvent indiquer l'itinéraire jusqu'au prochain nœud (dans le cas de plusieurs liaisons), à quoi se connecter ou quand envoyer un message à l'appareil cible. Chaque segment doit être constitué d'un octet de descripteur de segment qui spécifie le type de segment et de format ainsi que les informations de segment dont la taille dépend du type de segment / de format.

Un chemin a pour type de données EPATH (condensé ou bourré). Les deux types de chemin doivent être

- des chemins bourrés (le type de données indiqué est Padded EPATH);
- des chemins condensés (le type de données indiqué est Packed EPATH).

Les deux formats de chemin (bourré et condensé) ne doivent pas être interchangeables. L'utilisation de bourré ou de condensé doit être spécifiée en fonction du contexte d'utilisation.

Chaque segment d'un chemin bourré doit être un mot de 16 bits aligné. Si un octet de bourrage est requis pour obtenir l'alignement, le segment doit spécifier l'emplacement de l'octet de bourrage. Un chemin condensé ne doit pas contenir d'octets de bourrage.

Les types de segment possibles doivent être comme suit:

Port segment (segment de port) (où);

Logical segment (segment logique) (quoi);

Network segment (segment de réseau) (quand ou comment);

Symbolic segment (segment symbolique);

Data segment (segment de données).

4.1.9.2 Type de segment

Chaque segment du chemin doit contenir un type de segment / un octet de formatage pour indiquer comment interpréter le segment. Le type de segment / le format est contenu dans le premier octet du segment. Les bits du premier octet d'un segment de chemin doivent être numérotés de 0 à 7, où le bit 0 doit être le bit de poids le plus faible de l'octet. Les bits du type de segment sont définis à la Figure 10.

Type de segment			**Format de segment**				
7	6	5	4	3	2	1	0

0	0	0	Port segment (segment de port)
0	0	1	Logical segment (Segment logique)
0	1	0	Network segment (Segment de réseau)
0	1	1	Symbolic segment (Segment symbolique)
1	0	0	Data segment (Segment de données)
1	0	1	Data Type (Type de données, construit, voir 5.2.3)
1	1	0	Data Type (Type de données, élémentaire, voir 5.2.2)
1	1	1	Réservé pour usage futur

Figure 10 – Type de segment

La signification des bits de Segment Format repose sur le Type de segment spécifié

4.1.9.3 Segment de port

Le segment de port doit indiquer

- le port de communication par lequel quitter le nœud (exprimé sous la forme d'un nombre de 16 bits);
- l'adresse de liaison du prochain appareil sur le chemin.

La Figure 11 montre la structure globale du segment de port.

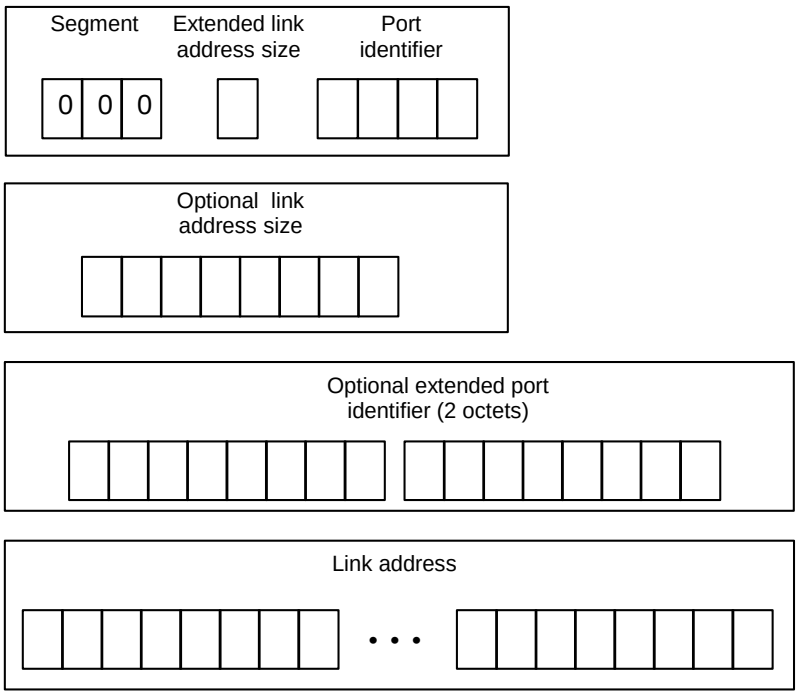
Les bits 5 à 7 du premier octet doivent être zéro pour indiquer le type de segment de port.

Le bit 4, bit de taille d'adresse de liaison étendue, doit être mis à 0 si l'adresse de liaison est d'un octet. Le bit 4 doit être mis à 1 si l'adresse de liaison est supérieure à un octet. Si l'adresse de liaison est supérieure à un octet, sa taille en octets doit être dans le deuxième octet du segment de port.

Les bits 0 à 3, l'identificateur de port, doivent indiquer le port par lequel quitter le nœud. L'identificateur de port doit spécifier un numéro de port ou un échappement à un identificateur de port étendu lorsque le module peut prendre en charge plus de 14 ports. Le numéro de port 0 doit être réservé. Le numéro de port 1 doit seulement être utilisé pour représenter un port de fond de panier. Si l'identificateur de port est 15, un champ de 16 bits appelé "identificateur de port étendu" doit être la partie suivante du segment de port (suivant la taille d'adresse de liaison facultative, le cas échéant); autrement, la valeur de l'identificateur de port doit être le numéro de port.

Le numéro de port doit être suivi d'une adresse de liaison dont le format dépend du type de réseau auquel l'identificateur de port fait référence. Si l'adresse de liaison est supérieure à un octet, elle doit être bourrée afin que le segment de port entier soit un nombre entier d'octets. L'octet de bourrage doit être mis à 0 et ne doit pas être Inclus dans la taille d'adresse de liaison.

NOTE 1 Pour les types de port communs, l'adresse de liaison est un octet unique. Les autres types de port, tels que l'encapsulation TCP/IP, peuvent utiliser une plus grande adresse de liaison (voir 4.3 et l'Article 11).



NOTE La représentation en bits se fait du bit de poids fort au bit de poids faible, de gauche à droite. La représentation en octets se fait de l'octet de poids faible à l'octet de poids fort, de haut en bas et de gauche à droite.

Légende

Anglais	Français
Segment	Segment
Extended link address size	Taille d'adresse de liaison étendue
Port identifier	Identificateur de port
Optional link address size	Taille d'adresse de liaison facultative
Optional extended port identifier (2 octets)	Identificateur de port étendu facultatif (2 octets)

Anglais	Français
Link address	Adresse de liaison

Figure 11 – Segment de port

Le format de l'Extended Port Identifier du Port Segment doit être utilisé seulement lorsqu'il y a plus de 14 ports possibles sur l'appareil. Le Port Segment doit toujours être condensé au format de Port Segment le plus petit possible par rapport aux champs facultatifs. Des exemples de segments de port possibles sont montrés dans le Tableau 156.

Tableau 156 – Exemples de segments de port possibles

Contenu du segment de port	Notes
[02][06]	Type de segment = port segment, numéro de port = 2, adresse de liaison = 6
[0F][12][00][01]	Type de segment = port segment. L'identificateur de port est 15, ce qui indique que le numéro de port est spécifié dans le champ de 16 bits suivant [12][00] (18 décimal). Adresse de liaison = 1
[15][0F][31][33][30][2E] [31][35][31][2E][31][33] [37][2E][31][30][35][00]	Type de segment = port segment. Adresse multioctet pour le port TCP 5; adresse de liaison 130.151.137.105 (adresse IP). L'adresse est définie comme une matrice de caractères d'une longueur de 15 octets. Le dernier octet du segment est un octet de bourrage.

La partie adresse de liaison d'un segment de chemin de connexion TCP/IP doit être codée au sein du segment de port sous la forme d'une chaîne de caractères ASCII. La chaîne doit avoir l'une des formes suivantes:

- adresse IP en notation avec point, par exemple "130.151.132.55" (voir IETF RFC 1117 pour le format des adresses IP);
- adresse IP en notation avec point, suivie d'un séparateur ":", suivi du numéro de port TCP à utiliser à l'adresse IP spécifiée;
- nom d'hôte, par exemple, "plc.type2.org". Le nom d'hôte doit être résolu par le biais d'une demande DNS à un serveur de nom (voir IETF RFC 1035 pour des informations relatives aux noms d'hôte et à la résolution des noms);
- nom d'hôte, suivi d'un séparateur ":", suivi du numéro de port TCP à utiliser auprès de l'hôte spécifié.

Le numéro de port doit être représenté soit en hexadécimal, soit en décimal. L'hexadécimal doit être indiqué par un "0x" de tête. Lorsqu'un numéro de port est spécifié, il doit être utilisé plutôt que le numéro de port normalisé utilisé pour le protocole d'encapsulation (0xAF12). Seul le port 0xAF12 est garanti comme étant disponible dans un appareil conforme à CP 2/2.

D'autres numéros de port TCP peuvent être mis en œuvre; cependant, la présente spécification ne fournit pas de mécanisme pour déterminer quels numéros de port TCP sont pris en charge par un appareil. L'utilisation d'autres numéros de port TCP est donc déconseillée.

NOTE 2 Le numéro de port TCP garanti, 0xAF12, a été réservé par l'Internet Assigned Numbers Authority (IANA) pour une utilisation par le protocole d'encapsulation.

Comme les segments de port doivent être alignés sur des mots, un octet de bourrage peut être requis à la fin de la chaîne. L'octet de bourrage doit être 0x00 et ne doit pas être compté dans le champ Optional Address Size du segment de port.

NOTE 3 Des exemples de segments de port pour TCP/IP sont montrés dans le Tableau 157 ci-dessous.

Tableau 157 – Exemples d'adresses de liaison TCP/IP

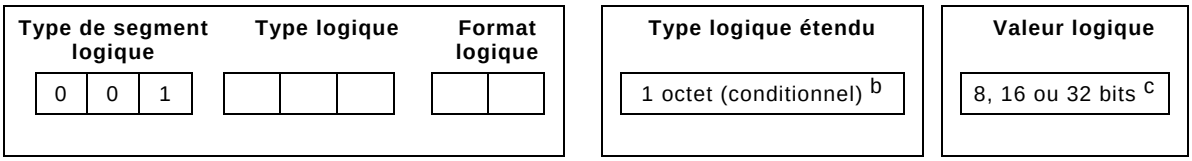
Segment de port	Adresse IP	Notes
[12][0D] [31][33][30][2E] [31][35][31][2E] [31][33][32][2E][31][00]	130.151.132.1	Adresse multioctet pour port 2, 13 octet string plus un octet de bourrage
[13][0C] [70][6C][63][2E] [74][79][70][32][2E] [6F][72][67]	plc.typ2.org	Adresse multioctet pour port 3, 12 octet string, sans octet de bourrage
[16][15] [31][33][30][2E] [31][35][31][2E] [31][33][32][2E] [35][35][3A] [30][78][33][32][31][30][00]	130.151.132.55:0x3210	Adresse multioctet pour port 6, 21 octet string plus un octet de bourrage
[15][11] [70][6C][63][2E] [74][79][70][32][2E] [6F][72][67][3A] [39][38][37][36][00]	plc.typ2.org:9876	Adresse multioctet pour port 5, 17 octet string plus un octet de bourrage

4.1.9.4 Logical segment (Segment logique)

4.1.9.4.1 Structure de Logical segment

Le segment logique sélectionne une entité adressable particulière au sein d'un appareil (par exemple, classe d'objets, instance d'objet et attribut d'objet). Lorsque le segment logique est inclus au sein d'un Packed Path, la valeur logique (Logical Value) doit être accolée à l'octet de type de segment sans insérer de bourrage entre eux.

La Figure 12 spécifie le codage d'un segment logique.



Class ID	0	0	0	0	0	Valeur logique 8 bits
Instance ID	0	0	1	0	1	Valeur logique 16 bits
Member ID	0	1	0	1	0	Valeur logique 32 bits
Point de connexion	0	1	1	1	1	Réservé pour usage futur
Identificateur d'attribut	1	0	0			
Special [a]	1	0	1			
Service ID [a]	1	1	0			
Logique étendu	1	1	1			

A Les types logiques Special et Service ID n'utilisent pas la définition d'adressage logique pour le format logique (Logical Format).

b Nécessaire quand le type logique est Extended Logical (logique étendu), (1 1 1), inexistant dans les autres cas. Les valeurs du type Extended Logical sont définies au Tableau 158.

c Dépend de la valeur de Logical Format.

Figure 12 – Codage de segment logique

Les types Extended Logical sont spécifiés au Tableau 158.

Tableau 158 – Type Extended Logical (logique étendu)

Valeur	Description
0	Réservé
1	indice de matrice
2	indice indirect de matrice
3	Indice de bit
4	indice indirect de matrice
5	Structure Member Handle
6	Structure Member Handle
7 à 255	Réservé

Les formats d'adresse logique 8 bits et 16 bits sont autorisés pour une utilisation avec tous les types logiques (Logical Type).

Le format d'adresse logique de 32 bits est uniquement admis pour les types logiquesInstance ID, Connection Point, Array Index, Bit Index, Structure Member Number et Structure Member Handle. Il est interdit pour tout autre Logical Type (réservé pour usage futur).

Le Class ID (identificateur de classe) spécifie un type d'objet au sein d'un appareil. L'Instance ID (identificateur d'instance) est un nombre entier attribué à un objet lorsqu'il est créé.. Instance 0, appelée l'instance de classe, spécifie la classe elle-même. Les attributs d'un objet sont une liste énumérée de valeurs visibles de l'extérieur. Attribute ID (identificateur d'attribut) spécifie quel attribut d'un objet est sélectionné. Les attributs complexes sont composés de membres. Member ID (identificateur de membre) spécifie le membre de l'attribut.

Quand un segment logique étendu est compris dans un chemin bourré (Padded Path) et que le format logique est de 8 bits, il faut ajouter un octet de bourrage après la valeur logique (Logical Value - les formats à 16 bits et 32 bits sont identiques au chemin condensé Packed Path) et lui donner pour valeur zéro. Pour tous les autres segments logiques, s'ils se trouvent dans un chemin bourré, il faut insérer dans les formats logiques de 16 bits et de 32 bits un bourrage entre l'octet de type de segment et la valeur logique (le format de 8 bits est identique au chemin condensé, Packed Path). L'octet de bourrage doit être mis à zéro.

Quand un segment logique étendu (Extended Logic Segment) est spécifié, l'octet Extended Logical Segment Value doit être placé entre l'octet Segment Type et la valeur logique (Logical Value).

Un segment logique étendu d'indice de matrice (Array Index Extended Logical Segment) est utilisé pour spécifier l'indice de base 0 dans l'élément précédent de l'EPATH qui doit être une matrice.

Un segment logique étendu d'indice de matrice indirecte (Indirect Array Index Extended Logical Segment) est utilisé pour spécifier le début d'un chemin d'application imbriqué qui donne un indice à base zéro dans l'élément précédent de l'EPATH qui doit être une matrice. La valeur logique de Indirect Array Index Logical Segment (Indirect Array Index Logical Segment Logical Value) est la taille du chemin d'application imbriqué, en mots de 16 bits.

Un segment logique étendu d'indice de bits (Bit Index Extended Logical Segment) est utilisé pour spécifier le nombre de bits à base 0 dans l'élément précédent de l'EPATH (il n'y a pas de restriction sur le type de l'élément précédent).

Un segment logique étendu d'indice indirect de bits (Indirect Bit Index Extended Logical Segment) est utilisé pour spécifier le début d'un chemin d'application imbriqué qui donne un nombre de bits à base 0 dans l'élément précédent de l'EPATH (il n'y a pas de restriction sur

le type de l'élément précédent). La valeur logique de Indirect Bit Index Logical Segment (Indirect Bit Index Logical Segment Logical Value) est la taille du chemin d'application imbriqué, en mots de 16 bits.

Un segment logique étendu de nombre de membre de la structure (Structure Member Number Extended Logical Segment) est utilisé pour spécifier le nombre de membres à base 1 dans l'élément précédent de l'EPATH qui doit être une structure. Le numéro de membre spécifie le membre de la structure, les membres de la structure sont comptés en commençant par 1.

Un segment logique étendu d'identification des membres de la structure (Structure Member Handle Extended Logical Segment) est utilisé pour spécifier l'identification des membres dans l'élément précédent de l'EPATH qui doit être une structure. L'identification est une valeur qui permet l'identification unique d'un membre et qui est fournie par l'appareil cible lors du rapport concernant les informations de type données construites à l'aide du codage formel, les éléments d'identification (Handles) étant spécifiés en 5.2.3.2.5.

Le type logique Connection Point fournit des capacités d'adressage supplémentaires au-delà de l'adresse d'objet normalisée Class ID/Instance ID/Attribute ID/Member ID. Les classes d'objets doivent définir quand et comment ce composant d'adressage est utilisé (par exemple, une sous-instance d'un objet Assembly).

NOTE La CEI 61158-5-2 décrit le modèle d'objet comportant la définition de classe, d'instance, d'attribut, de point de connexion et de membres d'une bibliothèque d'objets communs.

Le type logique Service ID a la définition suivante pour le format logique (Logical Format):

- 0 0 Segment "Service ID" de 8 bits (0x38)
- 0 1 Réservé pour usage futur (0x39)
- 1 0 Réservé pour usage futur (0x3A)
- 1 1 Réservé pour usage futur (0x3B)

Le type logique "Special" a la définition suivante pour le format logique (Logical Format):

- 0 0 Segment Electronic Key (clé électronique) (0x34)
- 0 1 Réservé pour usage futur (0x35)
- 1 0 Réservé pour usage futur (0x36)
- 1 1 Réservé pour usage futur (0x37)

4.1.9.4.2 Electronic Key Segment (Segment de clé électronique)

Le segment de clé électronique doit être utilisé pour vérifier/identifier un appareil. La clé peut être incluse comme premier segment logique dans un chemin de connexion et doit être vérifiée par le Message Router du nœud de réception par rapport aux valeurs contenues dans l'instance 1 de son objet Identity avant que ne soient effectuées toutes vérifications d'adresse supplémentaires. Plusieurs segments de clé peuvent apparaître sur un message multi-saut envoyé vers une liaison distante en passant par des routeurs intermédiaires. Le format du segment de clé électronique doit être tel que montré dans le Tableau 159.

Tableau 159 – Format du segment de clé électronique

Paramètre	Format	Valeur
segment_type	USINT	toujours 0x34
electronic_key_type	USINT	toujours 0x04
vendor_ID	UINT	ils correspondent aux valeurs dans l'instance 1 de l'objet Identity.
product_type	UINT	
product_code	UINT	

Paramètre	Format	Valeur
major_revision: 7	USINT	
compatible_match: 1	USINT	
minor_revision	USINT	

Le paramètre segment_type pour un segment de clé doit être 0x34. Le paramètre electronic_key_type doit déterminer le format d'un segment de clé spécifique et doit être mis à 0x04. Toutes les autres valeurs de electronic_key_type doivent être réservées.

Le paramètre vendor_ID doit spécifier le vendeur d'appareil ou zéro si aucun vendeur spécifique n'est requis.

Le paramètre product_type doit spécifier une classe de produits tels qu'une entrée numérique ou des sorties analogiques. Le paramètre product_type doit être ignoré s'il est mis à zéro. Le paramètre product_code doit être spécifique à un vendeur, chaque vendeur ayant un code unique. Le paramètre product_code doit être ignoré s'il est mis à zéro.

Les champs major_revision et compatible_match doivent être respectivement contenus dans les 7 bits de poids faible et le bit de poids le plus fort du même octet.

Le paramètre major_revision doit spécifier le niveau de fonctionnalité d'un appareil. Le paramètre major_revision doit être ignoré s'il est mis à zéro.

Le paramètre minor_revision doit spécifier la révision d'un appareil qui n'altère pas le comportement visible du réseau. Une valeur de zéro doit spécifier que l'émetteur de la demande accepte toute révision mineure.

Le paramètre compatible_match doit modifier la fonction correspondante de la clé.

Si le bit est effacé (= 0), tout paramètre non nul vendor_ID, product_type, product_code, major_revision et minor_revision doit correspondre exactement.

Si le bit est mis (= 1), l'appareil peut accepter la clé pour tout appareil qu'il peut émuler. Le niveau d'émulation doit être spécifique à chaque produit. Lorsque ce bit est mis (1), la valeur 0 n'est pas valide pour vendor_ID, product_code et major_revision. Si la valeur de vendor_ID, product_code ou major_revision est 0, le code d'erreur 0x04 (Erreur de segment de chemin) doit être retourné. La valeur de minor_revision doit être ignorée; n'importe quelle valeur peut être spécifiée. Il n'y a pas de restriction sur la valeur de product_type. Une valeur de product_type de 0 indique que l'appareil doit être compatible avec le type Generic Device.

NOTE 2 La CEI 61158-5-2 décrit l'objet Identity et les règles pour la mise à jour des champs de révision.

4.1.9.4.3 Exemples de segments logiques

Le Tableau 160 montre des exemples de segments logiques.

Tableau 160 – Exemples de segments logiques

Premier octet	Nom de segment logique	Exemples de formats bourrés (tels qu'émis)	Exemples de formats condensés (tels qu'émis)	Notes
0x20	8-bit class	[20][04]	[20][04]	classe 0x04 (objet Assembly)
0x21	16-bit class	[21][00][34][12]	[21][34][12]	classe 0x1234
0x24	8-bit instance	[24][04]	[24][04]	instance 0x04
0x25	16-bit instance	[25][00][34][12]	[25][34][12]	instance 0x1234
0x28	8-bit member	[28][04]	[28][04]	membre 0x04
0x29	16-bit member	[29][00][34][12]	[29][34][12]	membre 0x1234
0x2A	32-bit member			
0x2C	8-bit connection point	[2C][04]	[2C][04]	point de connexion 0x04
0x2D	16-bit connection point	[2D][00][34][12]	[2D][34][12]	point de connexion 0x1234
0x30	8-bit attribute	[30][04]	[30][04]	attribut 0x04
0x31	16-bit attribute	[31][00][34][12]	[31][34][12]	attribut 0x1234
0x34	electronic key	[34][04] [12][00] [EF][BE] [0E][0B] [83] [01]	[34][04] [12][00] [EF][BE] [0E][0B] [83] [01]	ID de vendeur = 0x0012 type de produit = 0xBEEF code produit = 0x0B0E révision majeure = 0x03 révision mineure = 0x01 accepter les clés compatibles = oui

4.1.9.5 Network segment (Segment de réseau)

4.1.9.5.1 Structure de Network segment

Le type de segment (premier octet) d'un segment de réseau doit se situer dans la plage de 0x40 à 0x5F comme montré dans le Tableau 161 (les bits de poids fort doivent être 010). Le segment de réseau doit être utilisé pour spécifier les paramètres réseau qui peuvent être requis par un nœud pour émettre un message sur un réseau. Le segment de réseau doit précéder immédiatement le segment de port de l'appareil auquel il s'applique. En d'autres termes, le segment de réseau doit être le premier élément dans le chemin que l'appareil reçoit.

Tableau 161 – Segments de réseau

Premier octet	Nom du segment de réseau
0x40	réservé
0x41	schedule segment
0x42	fixed tag segment
0x43	production inhibit time segment
0x44 à 0x4F	réservé
0x050	safety segment
0x051 à 0x05E	réservé
0x5F	extended network segment

4.1.9.5.2 Schedule Segment (Segment de programme)

Le type de segment du segment de réseau de programme doit être 0x41. Le segment de réseau de programme doit spécifier

- un multiplicateur qui, lorsqu'il est multiplié par le NUT (Network Update Time, temps de mise à jour réseau), donne le CM_API (intervalle réel entre paquets) du transport programmé;
- une phase qui détermine sur quelles valeurs du `Moderator.interval_count` émettre.

NOTE 1 La couche Liaison de données définit le Moderator.interval_count (voir la CEI 61158-4-2).

Ce segment doit être inclus dans le chemin vers un appareil afin que chaque nœud intermédiaire puisse programmer le temps d'émission de liaison pour le trafic programmé ultérieur. Le multiplicateur doit être l'une des valeurs suivantes, 1, 2, 4, 8, 16, 32, 64 ou 128. La phase doit être dans la plage 0 à (multiplicateur – 1) Le deuxième octet du segment de réseau de programme doit coder le multiplicateur et aussi la phase en les ajoutant ensemble.

NOTE 2 Si un transport produit tous les 64 NUT en commençant à `interval_count` = 17, la valeur codée est 17 + 64 = 81.

Une valeur codée de zéro doit spécifier que le transport n'a pas encore reçu la permission d'utiliser la priorité programmée. Si un nœud reçoit une demande pour une connexion programmée dans laquelle n'existe aucun segment de réseau de programme ou le segment de réseau de programme existe, mais la valeur codée est 0, le nœud doit retourner le statut général = 0x01 et le statut étendu = 0x0317.

NOTE 3 Un émetteur de connexion est autorisé à utiliser la priorité programmée par le truchement de l'objet Scheduling (CEI 61158-5-2).

Pour les nœuds intermédiaires, si seul le port d'entrée ou le port de sortie est connecté au sous-réseau CP 2/1, un seul segment de programme est alors nécessaire si la priorité est programmée. Le segment de programme s'applique au port connecté au sous-réseau CP 2/1.

Pour les nœuds intermédiaires, si le port d'entrée et le port de sortie sont à la fois connectés au sous-réseau CP 2/1, deux segments de programme sont alors nécessaires si la priorité est programmée. Le premier segment de programme s'applique au port d'entrée et le deuxième segment de programme s'applique au port de sortie.

Pour les nœuds cible, un seul segment de programme est nécessaire si le port d'entrée est connecté au sous-réseau CP 2/1 et que la priorité est programmée.

4.1.9.5.3 Fixed Tag Segment (Segment à étiquette fixe)

Le type de segment du segment de réseau à étiquette fixe doit être 0x42. Le segment de réseau à étiquette fixe doit spécifier l'étiquette fixe qui est à utiliser pour envoyer un message non connecté. Ce sous-type de segment doit précéder le segment de port dans le chemin. Si le segment à étiquette fixe est absent, l'étiquette fixe par défaut doit être utilisée.

NOTE Le segment à étiquette fixe peut être utilisé au sein du chemin du service Unconnected_Send (voir 0) du Connection Manager pour envoyer des demandes à l'objet Keeper par le truchement du Management UCMM (voir la CEI 61158-5-2).

4.1.9.5.4 Production Inhibit Time Segment (Segment de temps de neutralisation de production)

Le type de segment du segment de temps de neutralisation de production doit être 0x43. Le deuxième octet doit spécifier le temps de neutralisation de production, qui est le temps minimal, en millisecondes, entre des émissions successives de données connectées pour la connexion spécifiée (plage de 1 ms à 255 ms). Si le temps de neutralisation de production est 0, il ne doit pas y avoir de limite sur la vitesse d'envoi des données sur la connexion.

EXEMPLE Si un temps de neutralisation de production de 10 ms est spécifié, de nouvelles données doivent être envoyées au plus tôt 10 ms après les données précédentes.

Lorsque le segment de temps de neutralisation de production est utilisé, le RPI doit déterminer quand renvoyer les données. Si le segment de temps de neutralisation de

production est omis au cours de l'établissement d'une connexion, un temps de neutralisation de production par défaut égal à 1/4 du RPI doit être utilisé. Le RPI doit être supérieur ou égal au temps de neutralisation de production. Si le RPI est inférieur au temps de neutralisation de production, la réponse Forward_Open doit être retournée avec l'une des erreurs suivantes:

- le RPI est inférieur au temps de neutralisation de production (statut 0x01, statut étendu 0x11B): recommandé;
- le RPI n'est pas pris en charge (statut 0x01, statut étendu 0x0111): déconseillé.

Le segment de réseau de temps de neutralisation de production s'applique uniquement à l'appareil cible.

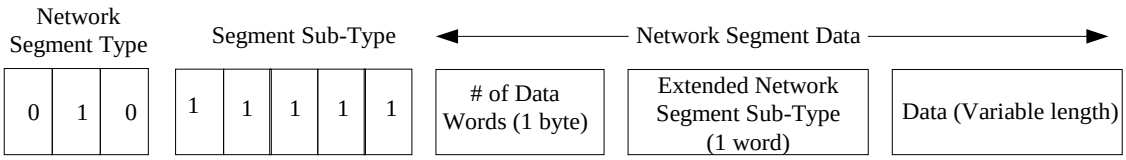
NOTE La méthode d'émission des APDU sur Ethernet-TCP/IP est spécifiée en 4.3 et à l'Article 11.

4.1.9.5.5 Safety Segment (Segment de sécurité)

Pour le format, voir la CEI 61784-3-2.

4.1.9.5.6 Extended Network Segment (Segment de réseau étendu)

Le segment de réseau étendu permet la définition de sous-types supplémentaires de segment de réseau. Le premier mot des données est le sous-type de segment de réseau étendu. La structure du segment de réseau étendu est montrée à la Figure 13.



Légende

Anglais	Français
Network Segment Type	Type de segment de réseau
Segment Sub-Type	Sous-type de segment
Network Segment Data	Données de segment de réseau
# of Data Words (1 byte)	Nombre de mots de données (1 octet)
Extended Network Segment Sub-Type (1 word)	Sous-type de segment de réseau étendu (1 mot)
Data (Variable length)	Données (longueur variable)

Figure 13 – Segment de réseau étendu

Chaque sous-type étendu définit les données qui suivent. Les sous-types étendus sont énumérés dans le Tableau 162.

Tableau 162 – Définitions des sous-types étendus

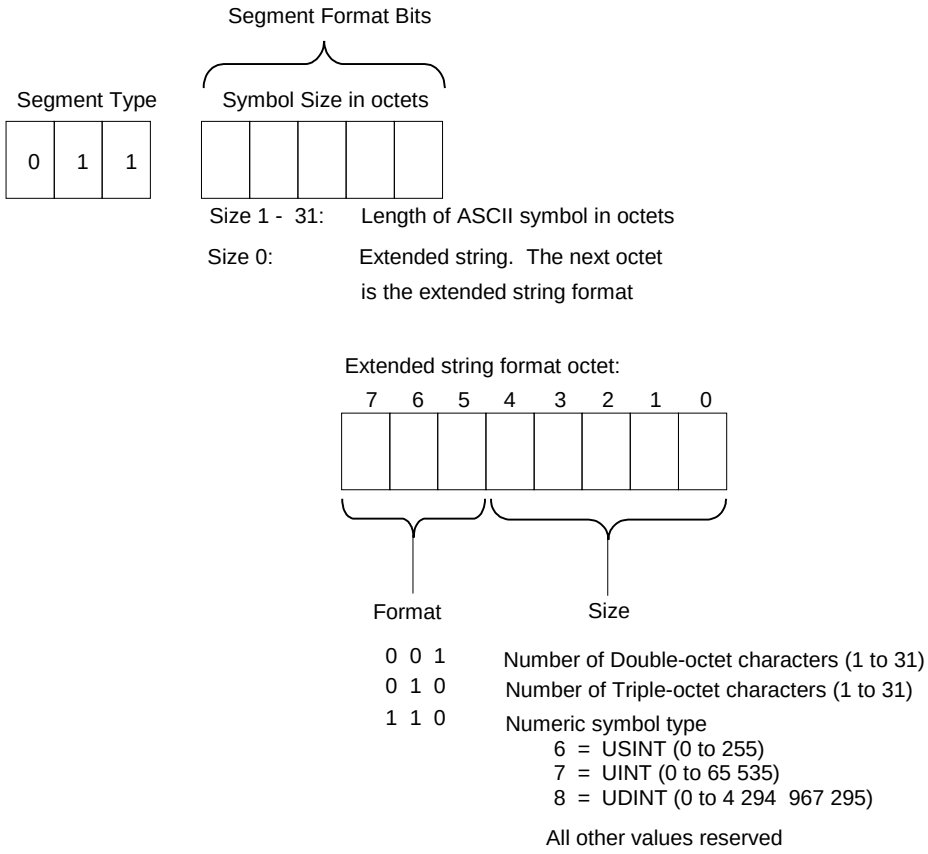
Valeur de sous-type étendu	Définition de sous-type étendu
0 à 32 767	Réservé pour usage futur
32 768 à 65 535	Spécifique à un vendeur [a]

[a] Les segments de réseau spécifiques à un vendeur doivent avoir pour seule cible le dernier appareil dans le paramètre Connection_Path des services de l'objet Connection Manager.

4.1.9.6 Segment symbolique

La plage de types de segment pour les segments symbolique doit être de 0x60 à 0x7F.

Le segment symbolique contient un symbole de chaîne internationale qui doit être interprété par l'appareil. Son format est spécifié à la Figure 14.



Légende

Anglais	Français
Symbol Size in octets	Taille de symbole en octets
Segment Format Bits	Bits de format segment
Segment Type	Type de segme,t
Size 1 - 31: Length of ASCII symbol in octets	Taille 1 - 31: Longueur du symbole ASCII en octets
Size 0: Extended string. The next octet is the extended string format	Taille 0: Chaîne étendue. L'octet suivant est le format de la chaîne étendue
Extended string format octet:	Octet de format de chaîne étendue:
Size	Taille
Number of Triple-octet characters (1 to 31)	Nombre de caractères en triple octet (1 à 31)
Number of Double-octet characters (1 to 31)	Nombre de caractères en double octet (1 à 31)
Numeric symbol type 6 = USINT (0 to 255) 7 = UINT (0 to 65 535) 8 = UDINT (0 to 4 294 967 295)	Type de symbole numérique 6 = USINT (0 à 255) 7 = UINT (0 à 65 535) 8 = UDINT (0 à 4 294 967 295)
All other values reserved	Toutes autres valeurs réservées

Figure 14 – Codage de segment symbolique

Les symboles de caractère peuvent contenir au plus 31 caractères.

Si le symbole est un symbole ASCII à double octet ou à triple octet, lors de la comparaison des segments la portion de symbole doit être traitée d'une manière non sensible à la casse. Par exemple, "LS101" et "ls101" doivent être considérés comme équivalents.

Les symboles numériques peuvent avoir 8, 16 ou 32 bits.

Quand un segment symbolique est utilisé dans un Padded EPATHn si la longueur du segment est impaire il faut ajouter un octet de bourrage 0 à la fin du segment. La taille ne doit pas comporter l'octet de bourrage.

Le Tableau 163 montre des exemples de segments symboliques (les valeurs sont hexadécimales).

Tableau 163 – Exemples de segments symboliques

Symbolic segment (Segment symbolique)	Notes
[65][LS101]	Symbole.ASCII Pourrait faire référence à un bit de la structure de données
[67][Line_23]	Symbole.ASCII Pourrait faire référence à un contrôleur dans une encoche
[68][Wire_off]	Symbole.ASCII Pourrait faire référence à un bit dans une structure de diagnostic
[60][22][1234][2345]	Symbole japonais
[60][C7][1234]	Symbole numérique de 16 bits
[60][C8][12345678]	Symbole numérique de 32 bits

4.1.9.7 Data segment (Segment de données)

4.1.9.7.1 Structure et objet du segment de données

Le segment de données fournit un mécanisme qui permet de livrer des données à une application. Il peut intervenir au cours de l'établissement de la connexion ou à tout autre moment défini par l'application.

Le type de segment (premier octet) d'un segment de données doit se situer dans la plage de 0x80 à 0x9F (les trois bits de poids fort doivent être 100). La présente norme définit seulement le format du simple segment de données avec un type de segment de 0x80 et du segment de symbole étendu ANSI (0x91). Tous les autres types de segment de données doivent être réservés (0x81 à 0x90, 0x92 à 0x9F).

4.1.9.7.2 Simple Data Segment (Simple segment de données)

Le type de segment du simple segment de données doit être 0x80. Le deuxième octet doit représenter le nombre de mots de 16 bits des données de longueur variable (jusqu'à 255). Les données de longueur variable doivent suivre le deuxième octet comme montré dans le Tableau 164. Un chemin peut contenir plusieurs simples segments de données.

Tableau 164 – Segment de données

Paramètre	Format	Valeur
segment_type	USINT	toujours 0x80
segment_size	USINT	taille de la matrice data[], en mots de 16 bits
data[]	UINT	

Le simple segment de données contient des valeurs de données telles que des paramètres destinés à l'application cible. Les données peuvent être des informations de configuration pour un objet, des paramètres complémentaires nécessaires pour l'objet ou tout autre ensemble d'informations spécifiques à l'objet. Le segment de données ne doit être interprété par aucun autre appareil dans le chemin et, ainsi, seules les applications émettrices et cibles ont besoin de s'accorder sur son contenu.

4.1.9.7.3 ANSI Extended Symbol Segment (Segment de symbole étendu ANSI)

Le type de segment du segment de symbole étendu ANSI doit être 0x91. Le deuxième octet doit représenter le nombre caractères (de 8 bits) dans le symbole. Le symbole de longueur variable doit suivre le deuxième octet comme montré dans le Tableau 165.

La partie caractère du segment de symbole étendu ANSI doit être traitée sans tenir compte de la casse. Par exemple, "start1" et "Start1" doivent être considérés comme équivalents.

Tableau 165 – Segment ANSI_Extended_Symbol

Paramètre	Format	Valeur
segment_type	USINT	toujours 0x91
symbol_size	USINT	taille de la matrice symbol[]
symbol[]	USINT	
pad	USINT	seulement présent si symbol_size est impair, toujours mis à zéro

Le paramètre `symbol_size` doit être la taille de la matrice `symbol[]` en octets et ne doit pas être zéro. Le paramètre `symbol_size` ne doit pas compter l'octet `pad` (bourrage) s'il est inclus.

4.1.9.8 Hiérarchie des définitions de segment

En général, la définition des règles relatives à l'utilisation des segments est définie dans la classe d'objets et/ou le profil d'appareil. Lorsque des segments symboliques et/ou logiques sont utilisés pour construire un chemin d'application, la définition de la forme de Backus Naur est:

application_path::= CHOICE {*Symbolic_application_path*, *Class_application_path*}

Symbolic_application_path::= *symbolic_segment* [Connection_Point] [*member_specification*] [*bit_specification*]

symbolic_segment::= CHOICE {Symbolic_Segment, ANSI_Extended_Symbol_Segment}

Class_application_path::= Class_ID [*item_specification*]

item_specification::= CHOICE {*attribute_specification*, *connection_point_specification*}

attribute_specification::= Instance ID [[Attribute ID] [*member_specification*] [*bit_specification*]]

connection_point_specification::= [Instance ID] Connection Point [*member_specification*] [*bit_specification*]

member_specification::= CHOICE {Member_ID, *extended_member_specification*}

extended_member_specification::= SEQUENCE of {Array_Index, *indirect_array_specification*, Structure_Member_Number, Structure_Member_Handle}

indirect_array_specification::= Indirect_Array_Index *application_path*

bit_specification::= CHOICE {Bit_Index, *indirect_index_bit_specification*}

indirect_bit_specification::= Indirect_Bit_Index *application_path*

A *Symbolic_application_path* shall resolve, at run time, to a *Class_application_path*.

An Indirect_Array_Index and Indirect_Bit_Index *application_path* shall evaluate to a positive integer.

La profondeur du chemin d'application doit seulement agir d'après le degré requis par son application.

EXEMPLE

Voici quelques exemples de chemins d'application valides:

Class ID,

Class ID, Instance ID

Class ID, Instance ID, Attribute ID

Class ID, Instance ID, Attribute ID, Member ID

Class ID, Connection Point

Class ID, Connection Point, Member ID

Class ID, Instance ID, Connection Point (voir contrainte a) ci-dessous)

Class ID, Instance ID, Connection Point, Member ID (voir contrainte a) ci-dessous)

Symbolic ID

Symbolic ID, Member ID

Symbolic ID, Connection Point (voir contrainte a) ci-dessous)

Symbolic ID, Connection Point, Member ID (voir contrainte a) ci-dessous)

Symbolic ID, Array Index, Structure Member Number, Array Index, Bit Index

Symbolic ID, Indirect_Array_Index, [Class_ID, Instance_ID, Attribute_ID], Structure_Member_Number, Array_Index, Bit_Index

Les contraintes suivantes s'appliquent:

c) Instance ID et Connection Point pour la Class ID 4 (Assembly Object) ne doivent pas être utilisés ensemble puisque pour l'objet Assembly ils sont définis comme étant identiques.

d) Si le Symbolic ID donne une Class ID de 4 (Assembly Object) et une Instance ID, le Symbolic ID et le Connection Point ne doivent pas être utilisés ensemble puisque pour l'objet Assembly Instance et Connection Point sont définis comme étant identiques.

4.1.9.9 Règles de compression des chemins codés

Lorsque plusieurs chemins codés sont concaténés, la limite entre les chemins se situe à l'endroit où l'on rencontre un segment appartenant à un niveau supérieur de la hiérarchie. Plusieurs chemins codés peuvent être compactés s'ils partagent les mêmes valeurs aux niveaux supérieurs de la hiérarchie. Les segments logiques étendus (Extended Logical) ne doivent pas être utilisés dans des chemins compressés. Lorsque l'on rencontre un segment appartenant au même niveau ou à un niveau supérieur (excepté le niveau le plus élevé) de la hiérarchie, les niveaux supérieurs précédents sont utilisés pour le chemin codé suivant.

Les exemples ci-dessous montrent plusieurs chemins codés en représentation complète et compacte.

EXEMPLE 1

Complète: Class A, Instance A, Attribute A, Class A, Instance A, Attribute B

Compacte: Class A, Instance A, Attribute A, Attribute B

EXEMPLE 2

Complète: Class A, Instance A, Attribute A, Class A, Instance B, Attribute A

Compacte: Class A, Instance A, Attribute A, Instance B, Attribute A

Le format compact spécial ci-dessous est aussi défini, mais seulement quand la classe n'est pas l'objet Assembly.

Complète: Class A, Instance B, Class A, Instance B, Connection Point C, Class A, Instance B, Connection Point D, Data Segment

Compacte: Class A, Instance B, Connection Point C, Connection Point D, Data Segment

Quand un chemin de connexion comprend un chemin de configuration ou un chemin d'E/S qui est une instance de l'objet Assembly (code de classe 4) sans attribut, l'attribut de données (attribut 3) est présupposé.

EXEMPLE

Un chemin codé dont le contenu 20 04 24 xx 24 yy 24 zz est suivi par un segment de données est décodé de la manière suivante:

20 04 24 xx 30 03 chemin de configuration

20 04 24 yy 30 03 chemin E/S de consommation

20 04 24 zz 30 03 chemin E/S de production

4.1.10 Codes de classe, d'attribut et de service

4.1.10.1 Plages de codes

4.1.10.1.1 Plages définies

Il doit y avoir trois catégories pour les plages d'adresse des Class ID, des Attribute ID et des Service Codes comme spécifié dans le Tableau 166.

Tableau 166 – Catégories d'adressage

Cette catégorie	Se réfère à
Open (Ouverts)	Une valeur qui a la même signification pour tous les réalisateurs. Tous les services et classes d'objets définis dans la CEI 61158-5-2 s'inscrivent dans cette catégorie.
Vendor-specific (Spécifique à un vendeur)	Une plage de valeurs spécifiques au vendeur d'un appareil. Elles sont utilisées par les vendeurs pour étendre leurs appareils au-delà des options *Open* disponibles. Un vendeur gère en interne l'utilisation de valeurs dans cette plage. S'applique aux classes, attributs et services d'objets.
Object-class specific (Spécifique à une classe d'objets)	Une plage de valeurs dont la signification est définie par une classe d'objets. Cette plage s'applique aux définitions de Service Code.
NOTE Les valeurs ouvertes (OPEN) sont attribuées par ODVA, Inc.	

Les valeurs de Class ID, Attribute ID et Service Code doivent être telles que définies du Tableau 167 au Tableau 169.

4.1.10.1.2 Plages d'ID de code de classe

Les valeurs de Class ID doivent être telles que montrées dans le Tableau 167. Le code de classe = 0x00 est réservé et ne doit pas être utilisé.

Tableau 167 – Plages d'ID de code de classe

Plage (en hex)	Plage (en décimal)	Signification	Quantité
00 à 63	01 à 99	Open (Ouverts)	99
64 à C7	100 à 199	Spécifique à un vendeur	100
C8 à 2FF	200 à 239	Réservé	40
F0 à 2FF	240 à 767	Open (Ouverts)	528
300 à 4FF	768 à 1 279	Spécifique à un vendeur	512
500 à FFFF	1 280 à 65 535	Réservé	64 256
NOTE Les plages réservées peuvent être définies par ODVA, Inc.			

4.1.10.1.3 Plages d'ID d'attribut

Les valeurs de l'Attribute ID doivent être telles que montrées dans le Tableau 168. L'ID d'attribut = 0x00 est réservé et ne doit pas être utilisé.

Tableau 168 – Plages d'ID d'attribut

Plage (en hex)	Plage (en décimal)	Signification	Quantité
00 à 63	00 à 99	Open (Ouverts)	100
64 à C7	100 à 199	Spécifique à un vendeur	100
C8 à FF	200 à 255	Réservé	56
100 à 2FF	240 à 767	Open (Ouverts)	512
300 à 4FF	768 à 1 279	Spécifique à un vendeur	512
500 à 8FF	1 280 à 2 303	Open (Ouverts)	1 024
900 à CFF	2 304 à 3 327	Spécifique à un vendeur	1 024
D00 à FFFF	3 328 à 65 535	Réservé	62 208
NOTE Les plages réservées peuvent être définies par ODVA, Inc.			

4.1.10.1.4 Plages de codes de service

Le **Service code** doit être une valeur hexadécimale unique attribuée à chaque service. Les valeurs de Service Code doivent être telles que montrées dans le Tableau 169. Le code de service = 0x00 est réservé et ne doit pas être utilisé. Les codes de service spécifiques à un objet doivent être uniques seulement au sein de la classe qui les définit.

Tableau 169 – Plages de codes de service

Plage (en hex)	Plage (en décimal)	Signification	Quantité
00	00	Réservé	1

Plage (en hex)	Plage (en décimal)	Signification	Quantité
01 à 31	01 à 49	Open (Ouverts). Ils sont appelés *Common Services* (services communs). Ils sont définis dans la CEI 61158-5-2	49
32 à 4A	50 à 74	Spécifique à un vendeur	25
4B à 63	75 à 99	Object Class Specific (Spécifiques à une classe d'objets)	25
64 à 7F	100 à 127	Réservé	28
80 à FF	128 à 255	Réservés pour des messages de réponse	128
NOTE Les plages réservées peuvent être définies par ODVA, Inc.			

4.1.10.2 Définitions des codes

4.1.10.2.1 Classes d'objets de communication

Le Tableau 170 définit les codes de classe pour les classes d'objets. Tous les autres codes de classe énumérés dans la plage "open" et non énumérés dans le Tableau 170 doivent être réservés. Les exigences pour ces objets sont définies dans la CEI 61158-5-2.

Tableau 170 – Codes de classe

Code de classe (en hex)	Code de classe (en décimal)	Nom d'objet
0x01	1	Objet Identity
0x02	2	Objet Message Router
0x03	3	Objet DeviceNet[a]
0x04	4	Objet Assembly
0x05	5	Objet Connection
0x06	6	Objet Connection Manager
0x0F	15	Objet Parameter
0x2B	43	Objet Acknowledge Handler
0x39	57	Objet Safety Supervisor[b]
0x3A	58	Objet Safety Validator[b]
0x42	66	Objet Motion Device Axis[c]
0x43	67	Objet Time Sync
0x47	71	Objet DLR[a]
0x48	72	Objet QoS[a]
0x4C	76	Objet SERCOS III Link[d]
0xF0	240	Objet ControlNet[a]
0xF1	241	Objet Keeper[a]
0xF2	242	Objet Scheduling[a]
0xF3	243	Objet Connection Configuration[a]
0xF4	244	Objet Port[a]
0xF5	245	Objet TCP/IP Interface[a]
0xF6	246	Objet Ethernet Link[a]

[a] Cette classe d'objets est partie intégrante de la gestion de système.

[b] Cette classe d'objet est spécifiée dans la CEI 61784-3-2.

[c] Cette classe d'objet est spécifiée dans la CEI 61800-7-202.

[d] Cette classe d'objet est spécifiée dans la CEI 61784-3-2 édition 3.0 (en cours de préparation).

4.1.10.2.2 Attributs de classe prédéfinis

Sept ID d'attribut de classe prédéfinis doivent être réservés et ces attributs de classe prédéfinis/réservés doivent avoir leurs définitions énumérées dans le Tableau 171. Comme ces attributs sont réservés, les numéros d'ID d'attribut de classe 1 à 7 doivent toujours être réservés. Par conséquent, si un attribut de classe est ajouté à une spécification de classe d'objets, il doit commencer par l'ID d'attribut n°8.

Tableau 171 – Attributs de classe réservés pour toutes les définitions de classe d'objets

Identificateur d'attribut	Nom	Type de données	Sémantique des valeurs
1	Revision	UINT	La valeur de départ attribuée à cet attribut est un (1). Si des mises à jour nécessitant une augmentation de cette valeur sont effectuées, la valeur de cet attribut augmente de un (1).
2	Max Instance	UINT ou UDINT	La définition de classe doit définir le type de données utilisé.
3	Number of Instances	UINT ou UDINT	La définition de classe doit définir le type de données utilisé.
4	Optional attribute list	STRUCT of	
	Number of attributes	UINT	
	Optional attributes	ARRAY of UINT	
5	Optional service list	STRUCT of	
	Number services	UINT	
	Optional services	ARRAY of UINT	
6	Maximum ID Number Class Attributes	UINT	
7	Maximum ID Number Instance Attributes	UINT	

4.1.10.2.3 Codes d'OM_Service

Les codes des services communs pour les objets de réseau de contrôle déterministe doivent être tels que montrés dans le Tableau 172.

NOTE Le Tableau 172 énumère les codes et noms des services communs alors que la CEI 61158-5-2 donne une description générale de chaque service.

Tableau 172 – Liste des services communs

Code de service commun	Nom de service commun
0x00	réservé
0x01	Get_Attribute_All
0x02	Set_Attribute_All
0x03	Get_Attribute_List
0x04	Set_Attribute_List
0x05	Reset
0x06	Start
0x07	Stop

Code de service commun	Nom de service commun
0x08	Create
0x09	Delete
0x0A à 0x0C	réservé
0x0D	Apply_Attributes
0x0E	Get_Attribute_Single
0x0F	réservé
0x10	Set_Attribute_Single
0x11	Find_Next_Object_Instance
0x12 à 0x14	réservé
0x15	Restore
0x16	Save
0x17	NOP (No operation, aucune opération)
0x18	Get_Member
0x19	Set_Member
0x1A	Insert_Member
0x1B	Remove_Member
0x1C	Group_Sync
0x1D à 0x31	réservé

Les codes des services spécifiques à un objet pour l'objet Message Router doivent être tels que montrés dans le Tableau 173.

Tableau 173 – Liste des services spécifiques à un objet Message Router

Code de service spécifique à un objet	Nom du service
0x4B	Symbolic_Translation

Les codes des services spécifiques à un objet pour l'objet Acknowledge Handler doivent être tels que montrés dans le Tableau 174.

Tableau 174 – Liste des services spécifiques à un objet Acknowledge Handler

Code de service spécifique à un objet	Nom du service
0x4B	Add_AckData_Path
0x4C	Remove_AckData_Path

Le code du service spécifique à un objet pour l'objet Parameter doit être tel que montré dans le Tableau 175.

Tableau 175 – Liste des services spécifiques à un objet Parameter

Code de service spécifique à un objet	Nom du service
0x4B	Get_Enum_String

4.1.10.2.4 Codes de CM_Service

Les codes des services spécifiques au Connection Manager doivent être tels que montrés dans le Tableau 176.

NOTE Le Tableau 176 énumère les codes et noms des services alors que la CEI 61158-5-2 donne une description générale de chaque service.

Tableau 176 – Services spécifiques à Connection Manager

Code de service	Nom du service
0x4E	Forward_Close
0x52	Unconnected_Send
0x54	Forward_Open
0x56	Get_Connection_Data
0x57	Search_Connection_Data
0x5A	Get_Connection_Owner
0x5B	Large_Forward_Open

4.1.10.2.5 Codes de CO_Service

Les codes des services spécifiques à l'objet Connection doivent être tels que montrés dans le Tableau 177.

NOTE Le Tableau 177 énumère les codes et noms des services alors que la CEI 61158-5-2 donne une description générale de chaque service.

Tableau 177 – Services spécifiques à un objet Connection

Code de service	Nom du service
0x4B	Connection_Bind
0x4C	Producing_Application_Lookup
0x4E	SafetyClose
0x54	SafetyOpen

4.1.10.3 Types d'appareil

Afin de permettre l'interopérabilité et l'interversion, un type d'appareil doit être utilisé pour identifier des appareils semblables qui

- montrent le même comportement;
- produisent et/ou consomme le même jeu de données;
- contiennent le même jeu d'attributs configurables.

La définition formelle de ces informations est connue sous le nom de Device Profile (profil d'appareil): tous les appareils dont le numéro de Device Type est identique doivent satisfaire aux exigences spécifiées dans le Device Profile pour ce type d'appareil.

Le "Device Type" (type d'appareil) est un attribut d'instance requis de l'objet Identity tel que décrit dans la CEI 61158-5-2.

Les types d'appareil doivent être soit définis publiquement, soit spécifiques à un vendeur. Le Tableau 178 révèle le plan de numérotation à utiliser pour numéroter les types d'appareil.

Tableau 178 – Numérotation des types d'appareil

Plage (en hex)	Type	Quantité
00 à 63	Défini publiquement – Réservé	100
64 à C7	Spécifique à un vendeur	100
C8 à FF	Réservé	56
100 à 2FF	Défini publiquement – Réservé	512
300 à 4FF	Spécifique à un vendeur	512
500 à FFFF	Réservé	64 256

Tous les types d'appareil définis publiquement doivent avoir la même signification pour tous les réalisateurs et sont réservés. Le type Generic Device (appareil générique) (numéro de type = 0x00) doit définir un appareil qui ne correspond à aucun des types d'appareil définis.

NOTE La définition réelle des types d'appareil définis publiquement et des Device Profile correspondants ne relève pas du domaine d'application de la présente norme. Ils sont définis par ODVA, Inc.

Les types d'appareil spécifiques à un vendeur peuvent être développés et les vendeurs peuvent ne pas les publier. Chaque vendeur doit maintenir ses propres types d'appareil spécifiques à un vendeur et sa propre allocation de numéros correspondante.

4.1.11 Codes d'erreur

4.1.11.1 Erreurs de CM

Les codes d'erreur sont retournés avec la réponse à une demande de service Connection Manager qui a abouti à une erreur. Les codes d'erreur doivent être utilisés pour aider à diagnostiquer le problème avec une demande de service. Le code d'erreur doit être divisé en un statut général de 8 bits et un ou plusieurs mots de 16 bits de statut étendu. Sauf spécification contraire, seul le premier mot de statut étendu doit être requis. Des mots supplémentaires de statut étendu peuvent être utilisés pour spécifier des informations supplémentaires spécifiques à un appareil. Tous les appareils qui produisent des messages doivent être capables de gérer plusieurs mots de statut étendu.

Le Tableau 179 donne un résumé des codes d'erreur disponibles.

Seul le type de nœud indiqué dans la colonne "Détecté par" doit retourner le code de statut général/étendu. La colonne "Détecté par" peut prendre les valeurs Emetteur, Routeur et/ou Cible. En cas de détection par un routeur ou un appareil cible, ces codes d'erreur seront contenus dans le paquet de réponse. En cas de détection par un émetteur, ces codes d'erreur peuvent être retournés à l'application via une interface d'objet (par exemple, l'objet Connection Configuration) ou une autre interface interne.

Tableau 179 – Codes d'erreur d'une demande de service Connection Manager

Statut général	Statut étendu	Détecté par	Explication et description
0x00			Service achevé avec succès
0x01	0x0000 à 0x00FF		Désuet
0x01	0x0100	Routeur Cible	CONNECTION IN USE OR DUPLICATE FORWARD OPEN (connexion en utilisation ou doublon de forward_open) Ce code de statut étendu doit être retourné lorsqu'un émetteur tente d'établir une connexion vers une cible avec laquelle l'émetteur peut avoir déjà établi une connexion (Forward_Open non nul/correspondant – voir 4.1.5.3.2.2).
0x01	0x0101 à 0x0102		Réservé
0x01	0x0103	Cible	TRANSPORT CLASS AND TRIGGER COMBINATION NOT SUPPORTED (combinaison de classe de transport et de déclencheur non prise en charge) Une combinaison de classe de transport et de déclencheur a été spécifiée alors qu'elle n'est pas prise en charge par l'application cible.
0x01	0x0104 à 0x0105		Réservé
0x01	0x0106	Cible	OWNERSHIP CONFLICT (conflit de propriété) La connexion ne peut pas être établie, car une autre connexion a alloué en exclusivité certaines des ressources requises pour cette connexion. Un exemple en serait que seule une connexion de propriétaire exclusif peut contrôler un point de sortie sur un module E/S. Si une seconde connexion de propriétaire exclusif (ou une connexion de propriétaire redondant) est tentée, cette erreur doit être retournée.
0x01	0x0107	Cible	TARGET CONNECTION NOT FOUND (connexion cible introuvable) Ce code de statut étendu doit être retourné en réponse à la demande Forward_Close lorsque la connexion à fermer est introuvable au nœud cible. Les routeurs ne doivent pas générer ce code de statut étendu. Si la connexion spécifiée est introuvable au nœud intermédiaire, la demande de fermeture doit encore être transmise par le chemin spécifié dans la demande Forward_Close.
0x01	0x0108	Routeur Cible	INVALID NETWORK CONNECTION PARAMETER (paramètre de connexion réseau non valide) Ce code de statut étendu doit être retourné comme résultat de la spécification d'un type de connexion, d'une priorité de connexion, d'un propriétaire redondant ou d'une connexion fixe/variable que l'appareil ne prend pas en charge. NOTE Ce code d'état étendu est "déconseillé". Il est particulièrement recommandé d'utiliser plutôt 0x011F, 0x0120, 0x0121, 0x0122, 0x0123, 0x0124, 0x0125 ou 0x0132
0x01	0x0109	Routeur Cible	INVALID CONNECTION SIZE (taille de connexion non valide) Ce code de statut étendu est retourné lorsque la cible ou le routeur ne prend pas en charge la taille de connexion spécifiée. Cela peut se produire au niveau d'une cible parce que la taille ne correspond pas à la taille requise pour une connexion de taille fixe. Cela pourrait se produire au niveau d'un routeur si la taille demandée est trop grande pour le réseau spécifié. Un mot de statut supplémentaire peut suivre en indiquant la taille de connexion maximale prise en charge par le nœud répondeur. Le mot de statut supplémentaire est requis lorsqu'il est émis en réponse à la demande Large_Forward_Open. NOTE Ce code d'état étendu est "déconseillé". Il est particulièrement recommandé d'utiliser plutôt 0x0126, 0x0127 ou 0x0128.
0x01	0x010A à 0x010F		Réservé
0x01	0x0110	Cible	TARGET FOR CONNECTION NOT CONFIGURED (cible pour la connexion non configurée) Ce code de statut étendu doit être retourné lorsqu'une connexion est demandée à une application-cible qui n'a pas été configurée et la demande de connexion ne contient pas un segment de données pour la configuration (voir 4.1.9.7).

<table>
<tr><th>Statut général</th><th>Statut étendu</th><th>Détecté par</th><th>Explication et description</th></tr>
<tr><td>0x01</td><td>0x0111</td><td>Routeur
Cible</td><td>RPI NOT SUPPORTED (RPI non pris en charge)

Ce code de statut étendu doit être retourné si le dispositif ne peut pas prendre en charge le RPI demandé dans le sens O⇒T ou T⇒O. Ce code de statut étendu peut également être utilisé si le multiplicateur de temporisation de connexion (connection time-out multiplier) produit une valeur de temporisation qui n'est pas prise en charge par l'appareil ou bien le temps de neutralisation de production n'est pas valide.

Il est fortement recommandé d'utiliser le statut étendu 0x0112 lorsque la ou les valeurs de RPI ne sont pas acceptables.

Il est déconseillé d'utiliser de code d'état étendu quand le multiplicateur de temporisation de connexion n'est pas pris en charge. Il est particulièrement recommandé d'utiliser plutôt 0x0133.

Il est déconseillé d'utiliser ce code d'état étendu quand le temps de neutralisation de production n'est pas valide. Il est particulièrement recommandé d'utiliser plutôt 0x011B.</td></tr>
<tr><td>0x01</td><td>0x0112</td><td>Routeur
Cible</td><td>RPI VALUE(S) NOT ACCEPTABLE (valeur(s) de RPI non acceptable(s))

Ce code de statut étendu doit être retourné lorsque la ou les valeurs de RPI figurant dans la demande Forward_Open se situent en dehors de la plage requise par l'application dans l'appareil cible ou que la cible produit un intervalle différent. La cible doit comporter des informations avec un ou des RPI acceptables. Pour cette erreur, le statut étendu se compose de 6 mots de 16 bits et est formaté comme suit:
<table>
<tr><th>Type de données</th><th>Valeur</th><th>Explication du champ</th></tr>
<tr><td>UINT</td><td>0x0112</td><td>Code de statut étendu</td></tr>
<tr><td>USINT</td><td>variable</td><td>Type de RPI émetteur vers cible acceptable (voir ci-dessous):

0 – le RPI spécifié dans Forward_Open était acceptable (la valeur du RPI émetteur vers cible est ignorée) [a]

1 – non spécifié (permet de suggérer un autre RPI, par exemple la valeur par défaut)

2 – RPI minimal acceptable (utilisé lorsque le RPI était trop court pour la plage)

3 – RPI maximal acceptable (utilisé lorsque le RPI était trop long pour la plage)

4 – RPI requis pour corriger une discordance (utilisé lorsque des données sont déjà consommées selon un intervalle différent)

5 à 255 – réservé</td></tr>
<tr><td>USINT</td><td>variable</td><td>Type de RPI cible vers émetteur acceptable (voir ci-dessous):

0 – le RPI spécifié dans Forward_Open était acceptable (la valeur du RPI cible vers émetteur est ignorée) [a]

1 – non spécifié (permet de suggérer un autre RPI, par exemple la valeur par défaut)

2 – RPI minimal acceptable (utilisé lorsque le RPI était trop court pour la plage)

3 – RPI maximal acceptable (utilisé lorsque le RPI était trop long pour la plage)

4 – RPI requis pour corriger une discordance (utilisé lorsque des données sont déjà produites selon un intervalle différent, en général en cas de multidiffusion)

5 à 255 – réservé</td></tr>
</table></td></tr>
</table>

Statut général	Statut étendu	Détecté par	Explication et description		
			UDINT	variable	Valeur du RPI émetteur vers cible qui se situe à l'intérieur de la plage acceptable pour l'application. Ce champ est défini de la même manière que le paramètre RPI dans la demande Forward_Open.
			UDINT	variable	Valeur du RPI cible vers émetteur qui se situe à l'intérieur de la plage acceptable pour l'application. Ce champ est défini de la même manière que le paramètre RPI dans la demande Forward_Open.
			[a] Les types émetteur vers cible et cible vers émetteur ne doivent pas avoir tous deux la valeur 0.		
0x01	0x0113	Emetteur Routeur Cible	OUT OF CONNECTIONS (à court de connexions) Le Connection Manager ne peut pas prendre d'autres connexions. Le nombre maximal de connexions prises en charge par le Connection Manager a déjà été créé.		
0x01	0x0114	Routeur Cible	VENDOR ID OR PRODUCT CODE MISMATCH (discordance d'ID de vendeur ou de code de produit) Le code de produit ou l'ID de vendeur spécifié dans le segment logique "clé électronique" ne concorde pas avec le code de produit ou l'ID de vendeur de l'appareil. Si le bit de compatibilité est mis, ce code de statut étendu est retourné lorsque l'appareil ne peut pas émuler l'ID de vendeur ou le code de produit spécifié.		
0x01	0x0115	Routeur Cible	DEVICE TYPE MISMATCH (discordance de type d'appareil) Le Device Type spécifié dans le segment logique "clé électronique" ne concorde pas avec le Device Type de l'appareil. Si le bit de compatibilité est mis, ce code de statut étendu est retourné lorsque l'appareil ne peut pas émuler le Device Type spécifié.		
0x01	0x0116	Routeur Cible	REVISION MISMATCH (discordance de révision) Les révisions majeure et mineure spécifiées dans le segment logique "clé électronique" ne correspondent pas à une révision valide de l'appareil. Si le bit de compatibilité est mis, ce code de statut étendu est retourné lorsque l'appareil ne peut pas émuler la révision majeure spécifiée.		
0x01	0x0117	Cible	INVALID PRODUCED OR CONSUMED APPLICATION PATH (chemin d'application produit ou consommé non valide) Le chemin d'application produit ou consommé spécifié dans le chemin de connexion ne correspond pas à un chemin d'application produit ou consommé valide au sein de l'application cible. Cette erreur pourrait également être retournée si un chemin d'application produit ou consommé était requis, mais n'était pas fourni par une demande de connexion. NOTE Ce code d'état étendu est "déconseillé". Il est particulièrement recommandé d'utiliser plutôt 0x012A, 0x012B ou 0x012F.		
0x01	0x0118	Cible	INVALID OR INCONSISTENT CONFIGURATION APPLICATION PATH (chemin d'application de configuration non valide ou incohérent) Un chemin d'application spécifié pour les données de configuration ne correspond pas à une application de configuration ou n'est pas cohérent avec les chemins d'application consommés ou produits. Par exemple, le chemin de connexion spécifie des données de configuration de type float (flottantes) alors que les chemins produits ou consommés spécifient des données de type Integer (entières). NOTE Ce code d'état étendu est "déconseillé". Il est particulièrement recommandé d'utiliser plutôt 0x0129 ou 0x012F.		
0x01	0x0119	Cible	NON-LISTEN ONLY CONNECTION NOT OPENED (connexion qui ne soit pas en écoute seule non ouverte) Le demande de connexion échoue parce qu'il n'y a pas de types actuellement ouverts de connexion qui ne soit pas en écoute seule. Voir la CEI 61158-5-2, 6.3.1.4.5, pour une description des types de connexion d'application. Le code de statut étendu doit être retourné lorsqu'une tentative est faite d'établir un type de connexion en écoute seule vers une cible qui n'a aucune connexion qui ne soit pas en écoute seule déjà établie.		

Statut général	Statut étendu	Détecté par	Explication et description
0x01	0x011A	Cible	TARGET OBJECT OUT OF CONNECTIONS (objet cible à court de connexions) Le nombre maximal de connexions prises en charge par cette instance de l'objet cible a été dépassé. Par exemple, le Connection Manager pourrait prendre en charge 20 connexions alors que l'objet cible peut seulement prendre en charge 10 connexions. A la 11e demande de connexion vers l'objet cible, ce code de statut étendu serait utilisé pour signifier que le nombre maximal de connexions existe déjà dans l'objet cible.
0x01	0x011B	Cible	RPI IS SMALLER THAN THE PRODUCTION INHIBIT TIME (le RPI est inférieur au temps de neutralisation de production) Le RPI dans le sens Cible vers Emetteur est plus petit que le temps de neutralisation de production dans le sens Cible vers Emetteur.
0x01	0x011C	Routeur Cible	TRANSPORT CLASS NOT SUPPORTED (classe de transport non prise en charge) La classe de transport demandée dans le paramètre Transport Type/Trigger n'est pas prise en charge.
0x01	0x011D	Routeur Cible	PRODUCTION TRIGGER NOT SUPPORTED (déclencheur de production non pris en charge) Le déclencheur de production demandé dans le paramètre Transport Type/Trigger n'est pas pris en charge.
0x01	0x011E	Cible	DIRECTION NOT SUPPORTED (direction non prise en charge) La direction demandée dans le paramètre Transport Type/Trigger n'est pas prise en charge.
0x01	0x011F	Cible	INVALID ORIGINATOR TO TARGET NETWORK CONNECTION FIXVAR (connexion fixe/variable au réseau émetteur vers cible non valide) Ce code de statut étendu doit être retourné comme résultat de la spécification d'un fanion fixe/variable O⇒T non pris en charge.
0x01	0x0120	Cible	INVALID TARGET TO ORIGINATOR NETWORK CONNECTION FIXVAR (connexion fixe/variable au réseau cible vers émetteur non valide) Ce code de statut étendu doit être retourné comme résultat de la spécification d'un fanion fixe/variable T⇒O non pris en charge.
0x01	0x0121	Cible	INVALID ORIGINATOR TO TARGET NETWORK CONNECTION PRIORITY (priorité de connexion au réseau émetteur vers cible non valide) Ce code de statut étendu doit être retourné comme résultat de la spécification d'un code de priorité O⇒T non pris en charge.
0x01	0x0122	Cible	INVALID TARGET TO ORIGINATOR NETWORK CONNECTION PRIORITY (priorité de connexion au réseau cible vers émetteur non valide) Ce code de statut étendu doit être retourné comme résultat de la spécification d'un code de priorité T⇒O non pris en charge.
0x01	0x0123	Cible	INVALID ORIGINATOR TO TARGET NETWORK CONNECTION TYPE (type de connexion au réseau émetteur vers cible non valide) Ce code de statut étendu doit être retourné comme résultat de la spécification d'un type de connexion O⇒T non pris en charge.
0x01	0x0124	Routeur Cible	INVALID TARGET TO ORIGINATOR NETWORK CONNECTION TYPE (type de connexion au réseau cible vers émetteur non valide) Ce code de statut étendu doit être retourné comme résultat de la spécification d'un type de connexion T⇒O non pris en charge.
0x01	0x0125	Routeur Cible	INVALID ORIGINATOR TO TARGET NETWORK CONNECTION REDUNDANT_OWNER (propriétaire de connexion au réseau émetteur vers cible redondant non valide) Ce code de statut étendu doit être retourné comme résultat de la spécification d'un fanion Redundant Owner O⇒T non pris en charge.

<table>
<tr><th>Statut général</th><th>Statut étendu</th><th>Détecté par</th><th>Explication et description</th></tr>
<tr><td>0x01</td><td>0x0126</td><td>Cible</td><td>INVALID CONFIGURATION SIZE (taille de configuration non valide)

Ce code de statut étendu est retourné lorsque l'appareil cible détermine que le segment de données fourni dans le paramètre Connection_Path ne contenait pas un nombre acceptable de mots de 16 bits pour le chemin d'application de configuration demandé.

Un mot de statut supplémentaire doit suivre en indiquant la taille de configuration maximale prise en charge.
<table>
<tr><th>Type de données</th><th>Valeur</th><th>Explication du champ</th></tr>
<tr><td>UINT</td><td>0x0126</td><td>Code de statut étendu</td></tr>
<tr><td>UINT</td><td>Size</td><td>Taille maximale en mots</td></tr>
</table></td></tr>
<tr><td>0x01</td><td>0x0127</td><td>Routeur

Cible</td><td>INVALID ORIGINATOR TO TARGET SIZE (taille émetteur vers cible non valide)

Ce code de statut étendu est retourné par la cible lorsque la taille de l'objet consommateur déclarée dans la demande Forward_Open et disponible sur la cible ne correspond pas à la taille déclarée dans le paramètre O⇒T Network Connection Parameter.

Ce code de statut étendu est retourné par un routeur lorsque ce dernier ne peut pas prendre en charge la taille demandée dans le paramètre O⇒T Network Connection Parameter.

Un mot de statut supplémentaire doit suivre en indiquant la taille émetteur vers cible maximale prise en charge.
<table>
<tr><th>Type de données</th><th>Valeur</th><th>Explication du champ</th></tr>
<tr><td>UINT</td><td>0x0127</td><td>Code de statut étendu</td></tr>
<tr><td>UINT</td><td>Size</td><td>Taille maximale en octets</td></tr>
</table></td></tr>
<tr><td>0x01</td><td>0x0128</td><td>Routeur

Cible</td><td>INVALID TARGET TO ORIGINATOR SIZE (taille cible vers émetteur non valide)

Ce code de statut étendu est retourné par la cible lorsque la taille de l'objet producteur déclarée dans la demande Forward_Open et disponible sur la cible ne correspond pas à la taille déclarée dans le paramètre T⇒O Network Connection Parameter.

Ce code de statut étendu est retourné par un routeur lorsque ce dernier ne peut pas prendre en charge la taille demandée dans le paramètre T⇒O Network Connection Parameter.

Un mot de statut supplémentaire doit suivre en indiquant la taille cible vers émetteur maximale prise en charge.
<table>
<tr><th>Type de données</th><th>Valeur</th><th>Explication du champ</th></tr>
<tr><td>UINT</td><td>0x0128</td><td>Code de statut étendu</td></tr>
<tr><td>UINT</td><td>Size</td><td>Taille maximale en octets</td></tr>
</table></td></tr>
<tr><td>0x01</td><td>0x0129</td><td>Cible</td><td>INVALID CONFIGURATION APPLICATION PATH (chemin d'application de configuration non valide)

Le chemin d'application de configuration spécifié dans le chemin de connexion ne correspond pas à un chemin d'application de configuration valide au sein de l'application cible. Cette erreur pourrait également être retournée si un chemin d'application de configuration était requis, mais n'était pas fourni par une demande de connexion.</td></tr>
<tr><td>0x01</td><td>0x012A</td><td>Cible</td><td>INVALID CONSUMING APPLICATION PATH (chemin d'application consommateur non valide)

Le chemin d'application consommé spécifié dans le chemin de connexion ne correspond pas à un chemin d'application consommé valide au sein de l'application cible. Cette erreur pourrait également être retournée si un chemin d'application consommé était requis, mais n'était pas fourni par une demande de connexion.</td></tr>
</table>

Statut général	Statut étendu	Détecté par	Explication et description
0x01	0x012B	Cible	INVALID PRODUCING APPLICATION PATH (chemin d'application producteur non valide) Le chemin d'application produit spécifié dans le chemin de connexion ne correspond pas à un chemin d'application produit valide au sein de l'application cible. Cette erreur pourrait également être retournée si un chemin d'application produit était requis, mais n'était pas fourni par une demande de connexion.
0x01	0x012C	Cible	CONFIGURATION SYMBOL DOES NOT EXIST (symbole de configuration inexistant) Le symbole de configuration n'existe pas. L'émetteur tente de se connecter à un nom d'étiquette de configuration, mais ce nom ne figure pas dans la liste des étiquettes définies sur la cible.
0x01	0x012D	Cible	CONSUMING SYMBOL DOES NOT EXIST (symbole consommateur inexistant) Le symbole consommateur n'existe pas. L'émetteur tente de se connecter à un nom d'étiquette consommatrice, mais ce nom ne figure pas dans la liste des étiquettes définies sur la cible.
0x01	0x012E	Cible	PRODUCING SYMBOL DOES NOT EXIST (symbole producteur inexistant) Le symbole producteur n'existe pas. L'émetteur tente de se connecter à un nom d'étiquette productrice, mais ce nom ne figure pas dans la liste des étiquettes définies sur la cible.
0x01	0x012F	Cible	INCONSISTENT APPLICATION PATH COMBINATION (combinaison de chemins d'application incohérente) La combinaison de chemins d'application de configuration et/ou de consommation et/ou de production spécifiée dans le chemin de connexion est incohérente.
0x01	0x0130	Cible	INCONSISTENT CONSUME DATA FORMAT (format de données consommées incohérent) Les informations présentes dans le segment de données ne sont pas compatibles avec le format des données consommées. Par exemple, les données de configuration spécifient des données de configuration de type float (flottantes) alors que le chemin consommé spécifie des données de type integer (entières).
0x01	0x0131	Cible	INCONSISTENT PRODUCE DATA FORMAT (format de données produites incohérent) Les informations présentes dans le segment de données ne sont pas compatibles avec le format des données produites. Par exemple, les données de configuration spécifient des données de configuration de type float (flottantes) alors que le chemin produit spécifie des données de type integer (entières).
0x01	0x0132	Cible	LA FONCTION NULL FORWARD OPEN N'EST PAS PRISE EN CHARGE La cible ne prend pas en charge la fonction demandée par Null Forward Open. La fonction demandée peut être "ping a device" (envoyer un ping à l'appareil), "configure a device's application" (configurer une application d'appareil) ou "reconfigure a target device's application" (reconfigurer une application d'appareil cible).
0x01	0x0133	Cible Routeur	LE MULTIPLICATEUR DE TEMPORISATION DE CONNEXION N'EST PAS ACCEPTABLE Ce code d'état étendu doit être retourné après qu'a été spécifiée une valeur de multiplicateur de temporisation de connexion qui est réservée ou qui produit une valeur de temporisation trop grande pour que l'appareil la prenne en charge.
0x01	0x0134 à 0x0202		Réservé
0x01	0x0203	Emetteur	CONNECTION TIMED OUT (connexion expirée) Ce code de statut étendu doit se produire lorsqu'une connexion expire.

Statut général	Statut étendu	Détecté par	Explication et description
0x01	0x0204	Emetteur Routeur	UNCONNECTED REQUEST TIMED OUT (demande non connectée expirée) L'erreur Unconnected Request Timed Out (demande non connectée expirée) doit se produire lorsque l'UCMM temporise avant qu'une réponse ne soit reçue. Cela peut se produire pour un service Unconnected_Send, Forward_Open ou Forward_Close. Cela signifie typiquement que l'UCMM a tenté un nombre de fois spécifique à une liaison en utilisant un temporisateur de répétitions de tentative spécifique à une liaison et n'a pas reçu un acquittement ou une réponse. Cela peut résulter d'un engorgement au nœud de destination ou peut résulter du fait qu'un nœud ne soit pas alimenté en énergie ou présent.
0x01	0x0205	Routeur	PARAMETER ERROR IN UNCONNECTED REQUEST SERVICE (erreur de paramètre dans le service de demande non connecté) Par exemple, cela doit être causé par une combinaison de Connection Tick Time (voir CEI 61158-5-2, 6.2.3.2.1.7) et de Connection time-out dans un service Unconnected_Send, Forward_Open ou Forward_Close qui n'est pas prise en charge par un nœud intermédiaire.
0x01	0x0206	Emetteur Routeur	MESSAGE TOO LARGE FOR UNCONNECTED_SEND SERVICE (message trop gros pour le service Unconnected_Send) Cela doit être apparaître lorsque le service Unconnected_Send est trop gros pour être envoyé sur un réseau.
0x01	0x0207	Emetteur Routeur	UNCONNECTED ACKNOWLEDGE WITHOUT RESPONSE (acquittement non connecté sans réponse) Le message a été envoyé par le service de messages non connecté et un acquittement a été reçu, mais le message de réponse de données n'a pas été reçu.
0x01	0x0208 à 0x0300		Réservé
0x01	0x0301	Emetteur Routeur Cible	NO BUFFER MEMORY AVAILABLE (pas de mémoire-tampon disponible) Ce code de statut étendu doit se produire lorsque la mémoire-tampon des connexions disponible est insuffisante dans l'appareil.
0x01	0x0302	Emetteur Routeur Cible	LINK TRANSMIT TIME NOT AVAILABLE FOR DATA (temps d'émission de liaison non disponible pour les données) Ce code de statut étendu doit être retourné par tout appareil du chemin qui est un producteur et ne peut pas allouer un temps d'émission de liaison suffisant pour la connexion sur sa liaison. Cela peut se produire seulement pour les connexions qui sont spécifiées comme étant de priorité programmée.
0x01	0x0303	Emetteur Routeur Cible	NO CONSUMED CONNECTION ID FILTER AVAILABLE (aucun filtre d'ID de connexion consommée disponible) Tout appareil du chemin qui contient un consommateur de liaisons pour la connexion et n'a pas un filtre disponible de consumed_connection_id disponible doit retourner ce code de statut étendu.
0x01	0x0304	Emetteur Routeur Cible	NOT CONFIGURED TO SEND SCHEDULED PRIORITY DATA (non configuré pour envoyer des données de priorité programmée) S'il lui est demandé d'établir une connexion qui spécifie une priorité programmée, tout appareil qui est incapable d'envoyer des données dans la partie programmée de l'intervalle de temps de mise à jour de réseau doit retourner ce code de statut étendu. Par exemple, sur CP 2/1, ce code doit être retourné par un nœud dont l'ID de MAC est supérieur au nœud programmé maximal (SMAX).
0x01	0x0305	Routeur	SCHEDULE SIGNATURE MISMATCH (discordance de signature de programme) Ce code de statut étendu doit être retourné lorsque les informations de programmation de connexion dans l'appareil émetteur ne sont pas cohérentes avec les informations de programmation de connexion sur le réseau cible.
0x01	0x0306	Routeur	SCHEDULE SIGNATURE VALIDATION NOT POSSIBLE (validation impossible de la signature de programme) Ce code de statut étendu doit être retourné lorsque les informations de programmation de connexion dans l'appareil émetteur ne peuvent pas être validées sur le réseau cible. Par exemple, sur CP 2/1, ce code doit être retourné lorsqu'il n'y a pas de Keeper dans l'état maître.

Statut général	Statut étendu	Détecté par	Explication et description
0x01	0x0307 à 0x0310		Réservé
0x01	0x0311	Emetteur Routeur	PORT NOT AVAILABLE (port non disponible) Un port spécifié dans un segment de port n'est pas disponible ou n'existe pas.
0x01	0x0312	Emetteur Routeur	LINK ADDRESS NOT VALID (adresse de liaison non valide) Adresse de liaison spécifiée dans un segment de port non valide Ce code de statut étendu est le résultat d'un segment de port qui spécifie une adresse de liaison qui n'est pas valide pour le type de réseau cible. Le code de statut étendu ne doit pas être utilisé pour des adresses de liaison qui sont valides pour le type de réseau cible, mais ne répondent pas.
0x01	0x0313 à 0x0314		Réservé
0x01	0x0315	Emetteur Routeur Cible	INVALID SEGMENT IN CONNECTION PATH (segment non valide dans le chemin de connexion) Type de segment ou valeur de segment non valide dans le chemin de connexion Ce code de statut étendu résulte de l'incapacité d'un appareil à décoder le chemin de connexion. Par exemple, cela pourrait être dû à un type de chemin non reconnu ou à un type de segment apparaissant intempestivement. Ce code de statut étendu doit seulement être utilisé lorsqu'aucun autre code de statut étendu plus spécifique fourni dans le présent tableau ne s'applique.
0x01	0x0316	Routeur Cible	FORWARD CLOSE SERVICE CONNECTION PATH MISMATCH (discordance du chemin de connexion du service Forward_Close) Le chemin de connexion dans le service Forward_Close ne concorde pas avec le chemin de connexion dans la connexion fermée. Ce code de statut étendu a été "déconseillé" car le service Forward_Close utilise Connection triad pour la mise en correspondance, mais n'utilise pas le chemin de connexion.
0x01	0x0317	Emetteur Routeur Cible	SCHEDULING NOT SPECIFIED (programmation non spécifiée) Le segment de réseau Schedule n'était pas présent ou bien la valeur codée dans le segment de réseau Schedule n'est pas valide (0).
0x01	0x0318	Emetteur Routeur	LINK ADDRESS TO SELF INVALID (adresse de liaison en boucle non valide) Dans certaines conditions (en fonction de l'appareil), une adresse de liaison dans le segment de port qui pointe vers le même appareil (bouclage vers soi) est non valide.
0x01	0x0319	Routeur Cible	SECONDARY RESOURCES UNAVAILABLE (ressources secondaires non disponibles) Dans un système redondant double châssis, une demande de connexion qui est faite au système primaire doit être dupliquée sur le système secondaire. Si le système secondaire est incapable de dupliquer la demande de connexion, ce code de statut étendu doit être retourné.
0x01	0x031A	Cible	RACK CONNECTION ALREADY ESTABLISHED (connexion de baie déjà établie) Une demande pour une connexion de module a été refusée parce qu'une partie des données correspondantes est déjà incluse ans une connexion de baie.
0x01	0x031B	Cible	MODULE CONNECTION ALREADY ESTABLISHED (connexion de module déjà établie) Une demande pour une connexion de baie a été refusée parce qu'une partie des données correspondantes est déjà incluse ans une connexion de module.
0x01	0x031C	Emetteur Routeur Cible	MISCELLANEOUS (divers) Ce statut étendu est retourné lorsqu'aucun autre code de statut étendu ne s'applique pour une erreur liée à la connexion.

Statut général	Statut étendu	Détecté par	Explication et description
0x01	0x031D	Cible	REDUNDANT CONNECTION MISMATCH (discordance de connexion redondante) Ce code de statut étendu doit être retourné quand les champs ci-après ne concordent pas lors d'une tentative d'établissement d'une connexion de propriétaire redondant au même chemin cible: O=>T_RPI; O=>T_connection_parameters; T=>O_RPI; T=>O_connection_parameters; xport_type_and_trigger.
0x01	0x031E	Cible	NO MORE USER CONFIGURABLE LINK CONSUMER RESOURCES AVAILABLE IN THE PRODUCING MODULE (plus aucune autre ressource de consommateur de liaison configurable par l'utilisateur disponible dans le module producteur) Une cible doit retourner ce statut étendu lorsque le nombre configuré de consommateurs pour une application productrice est déjà en cours d'utilisation.
0x01	0x031F	Cible	NO USER CONFIGURABLE LINK CONSUMER RESOURCES CONFIGURED IN THE PRODUCING MODULE (aucune ressource de consommateur de liaison configurable par l'utilisateur configurée dans le module producteur) Une cible doit retourner ce statut étendu lorsqu'il n'y a pas de consommateurs configurés à utiliser par une application productrice.
0x01	0x0320 à 0x07FF		Spécifique à un vendeur
0x01	0x0800	Emetteur Routeur	NETWORK LINK OFFLINE (liaison réseau hors ligne) La liaison réseau dans le chemin vers le module est hors ligne.
0x01	0x0801 à 0x080F		Réservé
0x01	0x0810	Cible	NO TARGET APPLICATION DATA AVAILABLE (aucune donnée d'application cible disponible) Ce code de statut étendu est retourné lorsque l'application cible n'a pas de données valides à produire pour la connexion demandée.
0x01	0x0811	Emetteur	NO ORIGINATOR APPLICATION DATA AVAILABLE (aucune donnée d'application d'émetteur disponible) Ce code de statut étendu est retourné lorsque l'application d'émetteur n'a pas de données valides à produire pour la connexion demandée.
0x01	0x0812	Emetteur Routeur	NODE ADDRESS HAS CHANGED SINCE THE NETWORK WAS SCHEDULED (l'adresse de nœud a changé depuis que le réseau a été programmé.) Un routeur sur un réseau programmé (CP 2/1 par exemple) a une adresse de nœud différente de la valeur configurée dans l'émetteur de la connexion.
0x01	0x0813	Routeur Cible	NOT CONFIGURED FOR OFF-SUBNET MULTICAST (pas configuré pour une multidiffusion hors sous-réseau) Une connexion de multidiffusion a été demandée entre un producteur et un consommateur qui se situent sur des sous-réseaux différents et le producteur n'est pas configuré pour une multidiffusion hors sous-réseau.
0x01	0x0814	Cible	INVALID PRODUCE/CONSUME DATA FORMAT (format non valide de données de production/consommation) Les informations dans le segment de données indiquent que le format des données produites et/ou consommées n'est pas valide. NOTE Ce code d'état étendu est "déconseillé". Il est particulièrement recommandé d'utiliser plutôt 0x0130 ou 0x0131.
0x01	0x0815 à 0xFCFF		Réservé
0x01	0x0FD00 à 0xFFFF		Réservé (ne doit pas être utilisé)

Statut général	Statut étendu	Détecté par	Explication et description
0x02	Vide	Emetteur Routeur	RESOURCE UNAVAILABLE FOR UNCONNECTED_SEND (ressource non disponible pour Unconnected_Send) L'appareil ne dispose pas des ressources nécessaires pour traiter entièrement la demande Unconnected_Send.
0x04	Vide	Routeur	PATH SEGMENT ERROR IN UNCONNECTED SEND (erreur de segment de chemin dans Unconnected_Send) Indique que le routeur CPF 2 a rencontré une erreur d'analyse lors de l'extraction de la demande Explicit Messaging des données de service de demande Unconnected Send.
0x09	Index vers élément	Cible	ERROR IN DATA SEGMENT (erreur dans le segment de données). Ce code de statut général doit être retourné lorsqu'il y a une erreur dans le segment de données dans un Forward_Open. Le statut étendu doit être l'index vers l'endroit où l'erreur a été rencontrée dans le segment de donnée (voir 4.1.9.7).
0x0C	Facultatif	Cible	OBJECT STATE ERROR (erreur d'état d'objet) Ce code de statut général doit être retourné lorsque l'état de l'objet cible de la connexion empêche que la demande de service soit gérée. Le statut étendu rend compte de l'état actuel de l'objet. Le statut étendu est facultatif. Par exemple, un objet (application) cible de la connexion peut avoir besoin d'être dans un mode d'édition avant que des attributs ne puissent être fixés. Cela est différent du rejet d'un service du fait de l'état de l'appareil.
0x10	Facultatif	Routeur Cible	DEVICE STATE ERROR (erreur d'état d'appareil) Ce code de statut général doit être retourné lorsque l'état de l'appareil empêche que la demande de service soit gérée. Le statut étendu rend compte de l'état actuel de l'appareil. Le statut étendu est facultatif. Par exemple, un dispositif de commande peut avoir un commutateur à clé qui, lorsqu'il est mis dans l'état "hard run" (marche dure), fait échouer des demandes de service à plusieurs objets (à savoir éditions de programme). Ce code de statut général serait alors retourné.
0x13	Aucun	Routeur Cible	NOT ENOUGH DATA (pas assez de données) Le service n'a pas fourni assez de données pour accomplir l'opération spécifiée.
0x15	Aucun	Routeur Cible	TOO MUCH DATA (trop de données) Le service a fourni plus de données qu'il n'en était prévu.

4.1.11.2 Erreurs d'OM

4.1.11.2.1 Format du statut général

Le statut général utilisé dans les primitives de service est spécifié dans le Tableau 180.

Tableau 180 – Codes de statut général

Code de statut	Nom	Description et signification du code de statut
0x00	Success (Succès)	Le service a été accompli avec succès par l'objet spécifié.
0x01	Connection failure (Echec de connexion)	Un service lié à la connexion a failli le long du chemin de connexion.
0x02	Resource unavailable (Ressource non disponible)	Les ressources nécessaires pour que l'objet exécute le service demandé n'étaient pas disponibles.
0x03	Invalid parameter value (Valeur de paramètre invalide)	Voir le code de statut 0x20, qui est la valeur préférentielle à utiliser pour cet état.
0x04	Path segment error (Erreur de segment de chemin)	L'identificateur de segment de chemin ou la syntaxe de segment n'était pas compris(e) par le nœud de traitement. Le traitement de chemin doit cesser en cas d'erreur de segment de chemin.

Code de statut	Nom	Description et signification du code de statut
0x05	Path destination unknown (Destination de chemin inconnue)	Le chemin fait référence à une classe, une instance, un élément de structure d'objet qui n'est pas connu(e) ou n'est pas contenu(e) dans le nœud de traitement. Le traitement de chemin doit cesser en cas d'erreur de destination inconnue.
0x06	Partial transfer (Transfert partiel)	Une partie seulement des données prévues a été transférée.
0x07	Connection lost (Connexion perdue)	La connexion de messagerie a été perdue.
0x08	Service not supported (Service non pris en charge)	Le service demandé n'était pas mis en œuvre ou n'était pas défini pour cette instance de classe ou d'objet.
0x09	Invalid attribute value (Valeur d'attribut non valide)	Données d'attribut invalides détectées
0x0A	Attribute list error (Erreur de liste d'attributs)	Un attribut dans la réponse Get_Attribute_List ou Set_Attribute_List a un statut non nul.
0x0B	Already in requested mode/state (Déjà dans le mode/état demandé)	L'objet est déjà dans le mode/état demandé par le service.
0x0C	Object state conflict (Conflit d'état d'objet)	L'objet ne peut pas accomplir le service demandé dans son mode/état actuel.
0x0D	Object already exists (L'objet existe déjà)	L'instance demandée de l'objet à créer existe déjà.
0x0E	Attribute not settable (Attribut non réglable)	Une demande de modifier un attribut non modifiable a été reçue.
0x0F	Privilege violation (Violation de privilège)	Une vérification de permission/privilège a échoué.
0x10	Device state conflict (Conflit d'état d'appareil)	Le mode/état courant de l'appareil interdit l'exécution du service demandé.
0x11	Response data too large (Données de réponse trop volumineuses)	Les données à émettre dans le tampon de réponses sont plus volumineuses que le tampon de réponses alloué.
0x12	Fragmentation of a primitive value (Fragmentation d'une valeur de primitive)	Le service spécifiait une opération qui va fragmenter une valeur de données de primitive, c'est-à-dire, une moitié de type de données REAL.
0x13	Not enough data (Pas assez de données)	Le service n'a pas fourni assez de données pour accomplir l'opération spécifiée.
0x14	Attribute not supported (Attribut non pris en charge)	L'attribut spécifié dans la demande n'est pas pris en charge.
0x15	Too much data (Trop de données)	Le service a fourni plus de données qu'il n'en était prévu.
0x16	Object does not exist (L'objet n'existe pas.)	L'objet spécifié n'existe pas dans l'appareil.
0x17	Service fragmentation sequence not in progress (Séquence de fragmentation de service pas en cours)	La séquence de fragmentation pour ce service n'est pas active actuellement pour ces données.
0x18	No stored attribute data (Aucune donnée d'attribut stockée)	Les données d'attribut de cet objet n'étaient pas sauvegardées préalablement au service demandé.
0x19	Store operation failure (Echec de l'opération de stockage)	Les données d'attribut de cet objet n'étaient pas sauvegardées en raison d'une défaillance au cours de la tentative.
0x1A	Routing failure, request packet too large (Echec d'acheminement, paquet de demande trop volumineux)	Le paquet de demande de service était trop volumineux pour l'émission sur un réseau dans le chemin vers la destination. L'appareil routeur était forcé d'abandonner le service.
0x1B	Routing failure, response packet too large (Echec d'acheminement, paquet de réponse trop volumineux)	Le paquet de réponse de service était trop volumineux pour l'émission sur un réseau dans le chemin en provenance de la destination. L'appareil routeur était forcé d'abandonner le service.

Code de statut	Nom	Description et signification du code de statut
0x1C	Missing attribute list entry data (Données d'entrée de liste d'attributs absentes)	Le service n'avait pas fourni un attribut dans la liste d'attributs qui était nécessaire au service pour avoir le comportement demandé.
0x1D	Invalid attribute value list (Liste de valeurs d'attribut non valide)	Le service retourne la liste d'attributs fournie avec les informations de statut pour les attributs qui n'étaient pas valides.
0x1E	Embedded service error (Erreur de service intégré)	Un service intégré a donné lieu à une erreur.
0x1F	Vendor specific error (Erreur spécifique à un vendeur)	Une erreur spécifique à un vendeur s'est produite. Le champ code de statut étendu de la réponse d'erreur définit l'erreur particulière rencontrée. Il convient de n'utiliser ce code de statut général que si aucun des codes de statut présentés dans ce tableau ou dans la définition de classe d'objets ne reflète précisément l'erreur.
0x20	Invalid parameter (Paramètre non valide)	Un paramètre associé à la demande n'était pas valide. Ce code est utilisé lorsqu'un paramètre ne satisfait pas aux exigences de la présente spécification et/ou aux exigences définies dans une spécification d'objet d'application.
0x21	Write once value or medium already written (Une valeur ou un support inscriptible une seule fois déjà inscrite)	Une tentative est faite d'écrire sur un support inscriptible une seule fois (par exemple, lecteur non réinscriptible (WORM "Write-Once-Read-Many times", PROM "Programmable Read Only Memory") qui a déjà eu une écriture ou de modifier une valeur qui ne peut pas être changée une fois établie
0x22	Invalid Response Received (Réponse non valide reçue)	Une réponse non valide est reçue (par exemple, le code de service de réponse ne concorde pas au code de service de demande ou bien le message de réponse est plus court que la taille minimale de réponse prévue). Ce code de statut peut servir pour d'autres causes de réponses non valides.
0x23	Buffer overflow (Dépassement de capacité du tampon)	Le message reçu est plus volumineux que ce que le tampon de réception peut gérer. Le message tout entier a été rejeté.
0x24	Message format error (Erreur de format de message)	Le format de message reçu n'est pas pris en charge par le serveur.
0x25	Key Failure in path (Défaillance de clé dans le chemin)	Le segment de clé qui était inclus comme premier segment dans le chemin ne concorde pas avec le module de destination. Le statut étendu doit indiquer quelle partie de la vérification de la clé a échoué (voir le Tableau 181).
0x26	Path Size Invalid (Taille de chemin non valide)	Soit la taille du chemin qui était envoyé avec la demande de service n'est pas suffisamment grande pour permettre l'acheminement de la demande jusqu'à un objet, soit trop de données d'acheminement étaient incluses.
0x27	Unexpected attribute in list (Attribut inattendu dans la liste)	Une tentative a été faite de fixer un attribut qui ne pouvait pas être réglé à ce stade.
0x28	Invalid Member ID (Identificateur de membre non valide)	L'ID de membre spécifié dans la demande n'existe pas dans la Classe/l'Instance/l'Attribut spécifié(e).
0x29	Member not settable (Membre non réglable)	Une demande de modifier un membre non modifiable a été reçue.
0x2A	Group 2 only server general failure (Défaillance générale de serveur de groupe 2 seulement)	Ce code de statut peut seulement être rapporté par des serveurs CP 2/3 de Groupe 2 seulement avec un espace de code de 4 Ko ou moins et seulement à la place d'un Service non pris en charge, d'un Attribut non pris en charge et d'un Attribut non réglable.
0x2B	Unknown Type 15 error (Erreur de Type 15 inconnu)	Un traducteur de Type 2 à 15 a reçu un code d'exception de Type 15 inconnu.
0x2C	Attribute not gettable (Attribut non récupérable)	Une demande de lire un attribut non lisible a été reçue.
0x2D	Instance non effaçable	L'instance d'objet demandée ne peut pas être effacée
0x2E à 0xCF	Réservé	Réservés pour des extensions futures

Code de statut	Nom	Description et signification du code de statut
0xD0 à 0xFF	Reserved for object class and service errors (Réservés pour des erreurs de service et de classe d'objets)	Cette plage de codes de statut doit être utilisée pour indiquer des erreurs spécifiques à une classe d'objets. Il convient de n'utiliser cette plage que si aucun des codes de statut présentés dans ce tableau ne reflète précisément l'erreur qui s'est produite.

NOTE La CEI 61158-5-2 contient plus de détail sur les codes de statut généraux des réponses de service pour chaque service commun.

4.1.11.2.2 Format du statut étendu

Le `MR_response_Header` contient un paramètre appelé `Extended_status[]` qui est une donnée de statut étendu étiquetée.

L'utilisation et la définition réelles de cet extended status dépend généralement de la classe d'objets ou du service: chaque classe d'objets définit ses propres valeurs et plages de valeurs de statut étendu (y compris celles qui sont spécifiques à un vendeur).

Le Tableau 181 spécifie le format du statut étendu d'un statut général "Key Failure in path" (Défaillance de clé dans le chemin) (0x25).

Tableau 181 – Code de statut étendu d'un statut général "Key Failure in path"

Code de statut général	Code de statut étendu	Nom de statut	Explication et description
0x25	0x0114	VENDOR ID OR PRODUCT CODE MISMATCH (discordance d'ID de vendeur ou de code de produit)	Le code de produit ou l'ID de vendeur spécifié dans le segment logique "clé électronique" ne concorde pas le code de produit ou l'ID de vendeur de l'appareil cible. Si le bit de compatibilité est mis, ce code de statut étendu est retourné lorsque l'appareil ne peut pas émuler l'ID de vendeur ou le code de produit spécifié.
0x25	0x0115	DEVICE TYPE MISMATCH (discordance de type d'appareil)	Le Device Type spécifié dans le segment logique "clé électronique" ne concorde pas avec le Device Type de l'appareil cible. Si le bit de compatibilité est mis, ce code de statut étendu est retourné lorsque l'appareil ne peut pas émuler le Device Type spécifié.
0x25	0x0116	REVISION MISMATCH (discordance de révision)	Les révisions majeure et mineure spécifiées dans le segment logique "clé électronique" ne correspondent pas à une révision valide de l'appareil cible. Si le bit de compatibilité est mis, ce code de statut étendu est retourné lorsque l'appareil ne peut pas émuler la révision majeure spécifiée.
NOTE Ces codes de statut étendu sont identiques à ceux définis pour le code de statut général 0x01 de Connection Manager.			

4.1.11.2.3 Format des statuts général et étendu spécifiques à un objet

4.1.11.2.3.1 Généralités

Le Paragraphe 4.1.11.2.3 spécifie le format de l'état spécifique étendu et général de l'objet.

4.1.11.2.3.2 Format du statut d'objet Identity

Le Tableau 182 spécifie le format de statut général et étendu spécifiques à un objet pour l'objet Identity.

Tableau 182 – Codes de statut d'objet Identity

Code de statut général	Codes de statut étendu associés de 8 bits	Nom de statut	Description du statut
0x00 à 0xCF		General status codes (Codes de statut général)	Définis en 4.1.11.2.1
	0x00 à 0xFE		Codes de statut étendu réservés
	0xF0 à 0xFE	Vendor specific (Spécifique à un vendeur)	Vendor specific extended status codes (Codes de statut étendu spécifique à un vendeur)
	0xFF		Utilisé avec tous les codes de statut général lorsque requis et aucun autre code de statut étendu n'est attribué.
0xD0		Hardware diagnostic (Diagnostic du matériel)	Conditions d'autotest et de diagnostic du matériel de l'appareil
	0x00		Réservé
	0x01		Erreur de somme de contrôle (ou de CRC) – Espace de code/ROM – Section de boot
	0x02		Erreur de somme de contrôle (ou de CRC) – Espace de code/ROM – Section d'application
	0x03		Erreur de somme de contrôle (ou de CRC) – Mémoire NV (flash/EEPROM)
	0x04		Mémoire non volatile (NV) non valide – Mauvaise configuration
	0x05		Mémoire non volatile (NV) non valide – Aucune configuration établie
	0x06		Mauvaise mémoire RAM – La mémoire RAM dans l'appareil était déterminée comme comportant des cellules inopérantes.
	0x07		Mauvaise mémoire ROM/Flash
	0x08		Mauvaise mémoire Flash/EEPROM (NV)
	0x09		Erreur de câble d'interconnexion / problème de chemin de signal
	0x0A		Problème de puissance – Surintensité
	0x0B		Problème de puissance – Surtension
	0x0C		Problème de puissance – Sous-tension
	0x0D		Problème de capteur interne
	0x0E		Défaut d'horloge système
	0x0F		La configuration matérielle ne concorde pas avec la configuration NV
	0x10		Chien de garde désactivé/au repos
	0x11		Temporisateur de chien garde expiré
	0x12		Surchauffe de l'appareil
	0x13		Température ambiante hors des limites de fonctionnement
	0x14 à 0xEF	(réservé)	Réservé
	0xF0 à 0xFE		Vendor specific extended status codes (Codes de statut étendu spécifique à un vendeur)
	0xFF		Utilisé avec tous les codes de statut général lorsque requis et aucun autre code de statut étendu n'est attribué.

Code de statut général	Codes de statut étendu associés de 8 bits	Nom de statut	Description du statut
0xD1		Device status/states (Statut/états de l'appareil)	Evénements et conditions de statut d'appareil
	0x01		Puissance appliquée
	0x02		Appareil réinitialisé
	0x03		Perte de puissance d'appareil
	0x04		Activated (Activé)
	0x05		Deactivated (Désactivé)
	0x06		Entrer dans l'état d'autotest
	0x07		Entrer dans l'état d'attente
	0x08		Entrer dans l'état opérationnel
	0x09		Défaut récupérable mineur et non spécifique détecté
	0x0A		Défaut irrécupérable mineur et non spécifique détecté
	0x0B		Défaut récupérable majeur et non spécifique détecté
	0x0C		Défaut irrécupérable majeur et non spécifique détecté
	0x0D		Défaut(s) corrigé(s)
	0x0E		Modification de Ccv
	0x0F		Intervalle Heartbeat modifié
	0x10 à 0xEF	(réservé)	
	0xF0 à 0xFE	Vendor specific (Spécifique à un vendeur)	Vendor specific (Spécifique à un vendeur)
	0xFF		Utilisé avec tous les codes de statut général lorsque requis et aucun autre code de statut étendu n'est attribué.
0xD2 à 0xEF		Object specific general status codes (Codes de statut général spécifique à un objet)	Réservé – Pas encore attribué
	0x00 à 0xFF	Réservé	
0xF0 à 0xFF		Vendor specific general status codes (Codes de statut général spécifique à un vendeur)	Une erreur spécifique à un vendeur s'est produite. Le champ code de statut étendu de la réponse d'erreur définit l'erreur particulière rencontrée. Il convient de n'utiliser ce code de statut général que si aucun des codes de statut présentés dans ce tableau ou dans la définition de classe d'objets ne reflète précisément l'erreur.
	0x00 à 0xFF	Vendor specific extended status codes (Codes de statut étendu spécifique à un vendeur)	Tous les codes de statut étendu sont disponibles pour une association avec chaque code de statut général spécifique à un vendeur.

4.2 Spécification de syntaxe abstraite des données

4.2.1 Spécification de format de transport

Les couches inférieures des architectures de systèmes ouverts sont concernées par le transport de données utilisateur parmi les unités fonctionnelles distribuées. Dans ces couches, les données utilisateur peuvent être considérées simplement comme une séquence d'octets. Cependant, des entités de couche application peuvent manipuler les valeurs de types de données tout à fait complexes. Pour assurer l'indépendance entre la couche application et les couches inférieures, les types de données peuvent être spécifiés dans une notation de syntaxe abstraite.

Le fait de compléter la syntaxe abstraite par un ou plusieurs algorithmes (appelés règles de codage) peut déterminer les valeurs des octets de couche inférieure qui transportent les valeurs de couche application. La combinaison de la syntaxe abstraite avec un seul jeu de règles de transfert produit une syntaxe de transfert spécifique.

4.2.2 Notation de syntaxe abstraite

Les définitions de type de données fournies dans la présente NORME doivent être écrites en Notation de syntaxe abstraite numéro un (ASN.1), telle que définie dans l'ISO/CEI 8824-1. Ces définitions de type doivent faire partie du module "Network DataTypes" de l'ASN.1. La déclaration de début ASN.1 indiquant que ces définitions se trouvent dans ce module est:

```
control network DataTypes DEFINITIONS::= BEGIN
```

et la déclaration de fermeture ASN.1 doit être le mot-clé "END".

Les définitions abstraites qui suivent doivent comprendre le jeu de types de données réseau de commande. En outre, une disposition est prise pour étendre ou dériver de nouveaux types de données basés sur des types définis existants et pour les inclure dans un "dictionnaire de types".

4.2.3 Spécification des données réseau de commande

La notation [typeId] pour les données de chaîne binaire structurées et de sous-plage énumérées directement dérivées doit signifier que l'étiquette doit être prise dans le champ "type" de la VariableDictionaryEntry correspondante.

```
Network Data::= CHOICE{ElementaryData, DerivedData}

ElementaryData::= CHOICE{
            BOOL,
            FixedLengthInteger,
            FixedLengthReal,
            AnyTime,
            AnyDate,
            AnyString,
            FixedLengthBitString,
            EPATH}

DerivedData::= CHOICE {
            DirectlyDerivedData,
            EnumeratedData,
            SubrangeData,
            StructuredBitStringData,
            ARRAY,
            STRUCT,
            FunctionBlockData}

DirectlyDerivedData::= [typeId] NetworkData

EnumeratedData::= [typeId] USINT

SubrangeData::= [typeId] FixedLengthInteger
```

```
StructuredBitStringData::= [typeId] FixedLengthBitString
FixedLengthInteger::= CHOICE {SignedInteger, UnsignedInteger}
SignedInteger::= CHOICE {SINT, INT, DINT, LINT}
UnsignedInteger::= CHOICE {USINT, UINT, UDINT, ULINT}
FixedLengthReal::= CHOICE {REAL, LREAL}
AnyTime::= CHOICE {ITIME,TIME, FTIME, LTIME}
AnyDate::= CHOICE {DATE, TIME_OF_DAY, DATE_AND_TIME}
AnyString::= CHOICE {STRING, STRING2}
FixedLengthBitString::= CHOICE {SWORD, WORD, DWORD, LWORD}
BOOL::= [PRIVATE 1] IMPLICIT BOOLEAN
SINT::= [PRIVATE 2] IMPLICIT OCTET STRING– 1 octet
INT::= [PRIVATE 3] IMPLICIT OCTET STRING– 2 octets
DINT::= [PRIVATE 4] IMPLICIT OCTET STRING– 4 octets
LINT::= [PRIVATE 5] IMPLICIT OCTET STRING– 8 octets
USINT::= [PRIVATE 6] IMPLICIT OCTET STRING– 1 octet
UINT::= [PRIVATE 7] IMPLICIT OCTET STRING– 2 octets
UDINT::= [PRIVATE 8] IMPLICIT OCTET STRING– 4 octets
ULINT::= [PRIVATE 9] IMPLICIT OCTET STRING– 8 octets
REAL::= [PRIVATE 10] IMPLICIT OCTET STRING– 4 octets
LREAL::= [PRIVATE 11] IMPLICIT OCTET STRING– 8 octets
STIME::= [PRIVATE 12] IMPLICIT DINT
DATE::= [PRIVATE 13] IMPLICIT UINT
TIME_OF_DAY::= [PRIVATE 14] IMPLICIT UDINT
DATE_AND_TIME::= [PRIVATE 15] IMPLICIT SEQUENCE {
 time_of_day  UDINT,
 date  UINT }
STRING::= [PRIVATE 16] IMPLICIT SEQUENCE {
            charcount      UINT,
            stringcontents OCTET STRING}  one octet per character
SWORD::= [PRIVATE 17] IMPLICIT OCTET STRING– 1 octet
WORD::= [PRIVATE 18] IMPLICIT OCTET STRING– 2 octets
DWORD::= [PRIVATE 19] IMPLICIT OCTET STRING– 4 octets
LWORD::= [PRIVATE 20] IMPLICIT OCTET STRING– 8 octets
STRING2::= [PRIVATE 21] IMPLICIT SEQUENCE {
            charcount      UINT,
            string2contents OCTET STRING} – 2 octets/ character
FTIME::= [PRIVATE 22] IMPLICIT DINT
LTIME::= [PRIVATE 23] IMPLICIT LINT
ITIME::= [PRIVATE 24] IMPLICIT INT
STRINGN::= [PRIVATE 25] IMPLICIT SEQUENCE {
            charsize    UINT,
            charcount   UINT,
            stringNcontents OCTET STRING} – N octets/ character
SHORT_STRING::= [PRIVATE 26] IMPLICIT SEQUENCE {
            charcount      USINT,
            stringcontents OCTET STRING}  one octet per character
```

```
TIME        ::=   [PRIVATE 27] IMPLICIT DINT

EPATH       ::=   [PRIVATE 28] IMPLICIT OCTET STRING  – IEC 61158-6-2

STRINGI     ::=   [PRIVATE 30] IMPLICIT SEQUENCE{
            Stringnum   USINT (number of strings)
            array of    STRUCT
            language1   USINT (first character from ISO 639-2/T)
            language2   USINT (second character from ISO 639-2/T)
            language3   USINT (third character from ISO 639-2/T)
            datatype    EPATH limited to the values 0xD0,0xD5,0xD9,and
0xDA)
            charset     UINT     from IANA MIB Printer Codes
(IETF RFC 1759))
            stringcontents    CHOICE OF (SHORT_STRING, STRING, STRING2,
                        or STRINGN) -- based on datatype field

ARRAY::= SEQUENCE OF NetworkData – Toutes du même type de base

STRUCT::= SEQUENCE OF NetworkData – Peuvent être de types différents

FunctionBlockData::= SET{
            inputs  [0] IMPLICIT STRUCT OPTIONAL,
            outputs     [1] IMPLICIT STRUCT OPTIONAL,
            controlInputs  [2] IMPLICIT STRUCT OPTIONAL,
            controlOutputs [3] IMPLICIT STRUCT OPTIONAL}
```

4.2.4 Spécification/dictionnaires de types de données

La définition d'un objet peut inclure du texte qui définit des attributs. Les attributs doivent recevoir un **Data Type** dans une spécification d'objet. Le Data Type peut être l'un de ceux définis dans la présente norme ou être une extension spécifique à un objet apportée à la présente norme. La définition doit fournir une spécification de type (**Type Specification**) pour les données et doit fournir une structure pour étendre ou dériver de nouveaux types de données sur la base de types définis existants.

```
Dictionary::= CHOICE {VariableDictionary, TypeDictionary}

VariableDictionary::= SEQUENCE OF VariableDictionaryEntry

VariableDictionaryEntry::= SEQUENCE{
            name  AnyString,
            id FixedLengthInteger,
            type  TypeID,
            ranges   SEQUENCE OF Subrange,– for arrays
            accessPrivilege   BOOL {READ_ONLY(0), READ_WRITE(1)}
TypeID::= OCTET STRING  – ASN.1 encoded tag value of the

 DataTypeSpecification module

Subrange::= SEQUENCE {
            minValue FixedLengthInteger,
            maxValue FixedLengthInteger}

TypeDictionary::= SEQUENCE OF TypeDictionaryEntry

TypeDictionaryEntry::= SEQUENCE {
            name  AnyString,
            type  TypeID,
            spec  DataTypeSpecification}
```

```
DataTypeSpecification::= CHOICE {
            alt      [PRIVATE 0] IMPLICIT AlternateTypeSpec,
            bool  [PRIVATE 1] IMPLICIT NULL,    BOOL
            sint  [PRIVATE 2] IMPLICIT NULL,    SINT
            int   [PRIVATE 3] IMPLICIT NULL,    INT
            dint  [PRIVATE 4] IMPLICIT NULL,    DINT
            lint  [PRIVATE 5] IMPLICIT NULL,    LINT
            usint [PRIVATE 6] IMPLICIT NULL,    USINT
            uint  [PRIVATE 7] IMPLICIT NULL,    UINT
            udint [PRIVATE 8] IMPLICIT NULL,    UDINT
            ulint [PRIVATE 9] IMPLICIT NULL,    ULINT
            real  [PRIVATE 10]   IMPLICIT NULL,    REAL
            lreal [PRIVATE 11]   IMPLICIT NULL,    LREAL
            stime [PRIVATE 12]   IMPLICIT NULL,    STIME
            date  [PRIVATE 13]   IMPLICIT NULL,    DATE
            tod   [PRIVATE 14]   IMPLICIT NULL,    TIME_OF_DAY
            dat   [PRIVATE 15]   IMPLICIT NULL,    DATE_AND_TIME
            str1  [PRIVATE 16]   IMPLICIT NULL,    STRING
            sword [PRIVATE 17]   IMPLICIT NULL,    SWORD
            word  [PRIVATE 18]   IMPLICIT NULL,    WORD
            dword [PRIVATE 19]   IMPLICIT NULL,    DWORD
            lword [PRIVATE 20]   IMPLICIT NULL,    LWORD
            str2  [PRIVATE 21]   IMPLICIT NULL,    STRING2
            ftime [PRIVATE 22]   IMPLICIT NULL,    FTIME
            ltime [PRIVATE 23]   IMPLICIT NULL,    LTIME
            itime [PRIVATE 24]   IMPLICIT NULL,    ITIME
            strN  [PRIVATE 25]   IMPLICIT NULL,    STRINGN
            shstr [PRIVATE 26]   IMPLICIT NULL,    SHORT_STRING
            time     [PRIVATE 27]   IMPLICIT NULL,    TIME
            epath    [PRIVATE 28]   IMPLICIT NULL,    EPATH
            strI     [PRIVATE 30]   IMPLICIT NULL, -- STRINGI
            constructedData   CHOICE {
               abbrvStruc  [0]   IMPLICIT AbbreviatedStrucTypeSpec,
               abbrvArr    [1]   IMPLICIT AbbreviatedArrayTypeSpec,
               frmlStruc   [2]   IMPLICIT FormalStrucTypeSpec,
               frmlArr     [3]   IMPLICIT FormalArrayTypeSpec,
               expBitStr   [4]   IMPLICIT ExpandedFixedLenBitStrTypeSpec,
               expStr1     [5]   IMPLICIT ExpandedStringTypeSpec,
               expStr2     [6]   IMPLICIT ExpandedString2TypeSpec
               frm1HandleStruc [7]  IMPLICIT  FormalHandleStrucTypeSpec}
}

AbbreviatedStrucTypeSpec::= UINT

AbbreviatedArrayTypeSpec::= DataTypeSpecification

FormalStrucTypeSpec::= SEQUENCE OF DataTypeSpecification

FormalHandleStrucTypeSpec::= SEQUENCE OF DataTypeHandleSpecification

DataTypeHandleSpecification::= DataTypeSpecification MemberHandle

MemberHandle::= USINT, UINT, UDINT
FormalArrayTypeSpec::= SEQUENCE {
            lowBound [0] IMPLICIT FixedLengthInteger,  Array Lower Bound
            highBound   [1] IMPLICIT FixedLengthInteger,  Array Upper Bound
            dataType    DataTypeSpecification }
```

```
ExpandedFixedLenBitStrTypeSpec::= SEQUENCE {
            bitStrType DataTypeSpecification     SWORD, WORD, DWORD, or
LWORD
            bitFields [7] IMPLICIT BitFieldDef}

BitFieldDef::= SEQUENCE OF {
            bitDef [2] IMPLICIT OCTET STRING}    Length is always 2 octets.
                                                 First octet contains starting
                                                 Bit Position. Trailing octet
                                                 contains the number of bits.

ExpandedStringTypeSpec::= UINT– Longueur de chaîne en octets

ExpandedString2TypeSpec::= UINT– Longueur de chaîne en octets
AlternateTypeSpec::= CHOICE {
            directlyDerivedTypeSpec [0]   IMPLICIT    TypeID,
            subrangeTypeSpec        [1]   IMPLICIT    SubrangeTypeSpec ,
            enumeratedTypeSpec      [2]   IMPLICIT    EnumeratedTypeSpec,
            fbTypeSpec              [3]   IMPLICIT    FBTypeSpec}

SubrangeTypeSpec::= SEQUENCE{
            baseType    TypeID,      NOTE  minValue and maxValue
            minValue    FixedLengthInteger,   shall be within the range
            maxValue    FixedLengthInteger}   of baseType values

EnumeratedTypeSpec::= SEQUENCE OF AnyString

BitNameDefintion::= SEQUENCE {
            bitName  AnyString,
            bitNumber   USINT}
FBTypeSpec::= SET{
            inputs         [0] IMPLICIT FbtElementSpec OPTIONAL,
            outputs        [1] IMPLICIT FbtElementTypeSpec OPTIONAL,
            controlInputs  [2] IMPLICIT FbtElementTypeSpec OPTIONAL,
            controlOutputs [3] IMPLICIT FbtElementTypeSpec OPTIONAL}

FbtElementTypeSpec::= SEQUENCE OF ElementSpec

ElementSpec::= SEQUENCE {
            name     AnyString,
            typespec    ElementTypeSpec}

ElementTypeSpec::= CHOICE {
            [0] IMPLICIT TypeID,
            [1] IMPLICIT SubrangeTypeSpec,
            [2] IMPLICIT EnumeratedTypeSpec,
            [3] IMPLICIT FormalArrayTypeSpec,
            [4] IMPLICIT ExpandedStringTypeSpec,
            [5] IMPLICIT ExpandedString2TypeSpec}
```

La déclaration END qui suit doit terminer le module ASN.1 ouvert en 4.1.

END.

4.3 Syntaxe abstraite d'encapsulation

4.3.1 Protocole d'encapsulation

4.3.1.1 Généralités

Afin d'envoyer des TPDU sur TCP/IP, un protocole d'encapsulation est requis. Le Paragraphe 4.3.1 définit les exigences relatives au protocole d'encapsulation. Le protocole d'encapsulation est un protocole générique qui peut être utilisé pour transporter des données autres que des TPDU de Type 2.

Le protocole d'encapsulation définit un numéro de port TCP réservé qui doit être pris en charge par tous les appareils de Type 2 (0xAF12). Tous les appareils de Type 2 doivent accepter au moins deux connexions TCP sur le port TCP numéro 0xAF12. Une fois la connexion TCP au port TCP numéro 0xAF12 établie, toutes les données envoyées via le flux TCP doivent utiliser le format spécifié en 4.3.1.2 et en 4.3.1.3.

NOTE TCP est un protocole basé sur le flux. Il peut envoyer des paquets IP de pratiquement n'importe quelle longueur. Par exemple, si deux messages encapsulés dos à dos sont transmis à une pile TCP/IP, la pile TCP/IP peut placer les deux messages encapsulés dans une même trame Ethernet ou placer la moitié du premier message dans la première trame Ethernet et le reste dans la trame Ethernet suivante.

Le protocole d'encapsulation définit également un numéro de port UDP réservé qui doit être pris en charge par tous les appareils EtherNet/IP (0xAF12). Tous les appareils doivent accepter des paquets UDP sur le port UDP numéro 0xAF12. Lorsque le protocole UDP est utilisé pour envoyer un message encapsulé, l'intégralité de ce message doit être envoyée dans un seul paquet UDP. Un seul message encapsulé doit être présent dans chaque paquet UDP destiné au port UDP 0xAF12.

Certains messages encapsulés doivent être envoyés uniquement via TCP. D'autres peuvent être envoyés soit via UDP, soit via TCP. Pour obtenir des détails sur les commandes limitées à TCP, voir le Tableau 184.

4.3.1.2 Messages d'encapsulation

Tous les messages d'encapsulation envoyés via le port TCP ou le port UDP 0xAF12 doivent être constitués d'un en-tête de longueur fixe de 24 octets suivi d'une partie "données" facultative. La longueur totale des messages d'encapsulation (en-tête compris) doit être limitée à 65 535 octets. La longueur des messages d'encapsulation ne doit pas neutraliser les restrictions de longueur imposées par le protocole encapsulé. Sa structure doit être telle que montrée à la Figure 15, où le haut du diagramme indique la partie du message envoyé en premier sur la ligne.

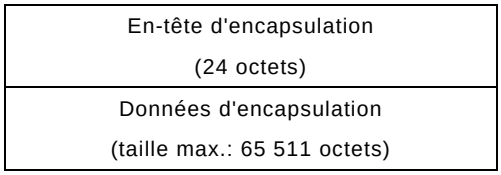


Figure 15 – Message d'encapsulation

La portion de message où se trouvent les données d'encapsulation est seulement nécessaire à certaines commandes.

4.3.1.3 En-tête d'encapsulation

L'en-tête d'encapsulation doit être tel que montré dans le Tableau 183.

Tableau 183 – En-tête d'encapsulation

Nom de champ	Format	Description
Command	UINT	Commande d'encapsulation
Length	UINT	Longueur, en octets, de la partie "données" spécifique à la commande du message, c'est-à-dire le nombre d'octets suivant l'en-tête
Session handle	UDINT	Identification de session (dépend de l'application)
Status	UDINT	Code de statut
Sender context	ARRAY of 8 USHORT	Informations pertinentes seulement pour l'expéditeur d'une commande d'encapsulation

Nom de champ	Format	Description
Options	UDINT	Fanions d'options

Les champs entiers multioctets dans les messages d'encapsulation doivent être émis comme spécifié en 4.3.3.4 (c'est-à-dire selon l'ordre d'octets little endian).

NOTE 1 Cet ordre d'octets est différent de celui utilisé dans les protocoles de réseau Internet normalisés, qui est big endian.

Bien que l'en-tête ne contienne aucune information explicite pour distinguer entre une demande et une réponse, ces informations doivent être déterminées de l'une de deux façons:

- de façon implicite, par la commande et le contexte dans lequel le message est généré. (Par exemple, dans le cas de la commande RegisterSession, la demande est générée par un émetteur et la cible génère la réponse);
- de façon explicite, par le contenu d'un paquet de protocole encapsulé dans la partie "données" du message.

NOTE 2 L'établissement d'une session est défini en 11.8. Une session crée une connexion TCP/IP entre l'émetteur et la cible, sur laquelle des commandes encapsulées peuvent être envoyées. Les connexions TCP/IP étant formées d'un train d'octets, l'en-tête d'encapsulation est ajouté au début de chaque paquet encapsulé de sorte que l'appareil récepteur puisse identifier le début et la fin de chaque paquet.

4.3.1.4 Champ Command

L'allocation de codes de commande doit être telle que montrée dans le Tableau 184.

Tableau 184 – Codes de commande d'encapsulation

Code de Command	Nom	Utilisation
0x0000	NOP	Peut être envoyé uniquement via TCP
0x0001 à 0x0003	Réservés pour usage hérité[a]	
0x0004	ListServices	Peut être envoyé soit via UDP, soit via TCP
0x0005	Réservés pour usage hérité[a]	
0x0006 à 0x0062	Réservés pour des extensions futures[b]	
0x0063	ListIdentity	Peut être envoyé soit via UDP, soit via TCP
0x0064	ListInterfaces	Facultatif (peut être envoyé soit via UDP, soit via TCP)
0x0065	RegisterSession	Peut être envoyé uniquement via TCP
0x0066	UnRegisterSession	Peut être envoyé uniquement via TCP
0x0067 à 0x006E	Réservés pour usage hérité[a]	
0x006F	SendRRData	Peut être envoyé uniquement via TCP
0x0070	SendUnitData;	Peut être envoyé uniquement via TCP
0x0071	Réservés pour usage hérité[a]	
0x0072	IndicateStatus	Facultatif (peut être envoyé uniquement via TCP)
0x0073	Cancel	Facultatif (peut être envoyé uniquement via TCP)
0x0074 à 0x00C7	Réservés pour usage hérité[a]	
0x00C8 à 0xFFFF	Réservés pour des extensions futures[b]	

[a] La mention "Réservé pour usage hérité" désigne des commandes qui ont été définies avant la publication de la présente norme. Leur comportement n'est pas défini dans la présente norme. Les appareils ne doivent pas mettre en œuvre ces commandes s'il n'a pas été pris connaissance de leur usage hérité. Les appareils qui ne prennent pas en charge ces commandes doivent retourner le code de statut d'encapsulation 0x0001.

Code de Command	Nom	Utilisation
b Les commandes portant la mention "Réservé pour des extensions futures" ne doivent pas être utilisées.		

Un appareil doit accepter des commandes qu'il prend effectivement en charge sans interrompre la session ou la connexion TCP sous-jacente. Un code de statut indiquant qu'une commande non prise en charge a été reçue doit être retourné à l'expéditeur du message (voir 4.3.1.7).

4.3.1.5 Champ Length

Le champ Length (longueur) dans l'en-tête doit spécifier la taille en octets de la partie "données" du message. Le champ doit contenir zéro pour les messages qui ne contiennent pas de données. La longueur totale d'un message doit être la somme du nombre contenu dans le champ Length plus la taille de 24 octets de l'en-tête d'encapsulation.

Le message d'encapsulation complet doit être lu sur la connexion TCP/IP même si la longueur n'est pas valide pour une commande particulière ou dépasse la capacité des tampons internes de l'hôte. Les données dépassant la capacité des tampons internes peuvent être rejetées, mais le message d'encapsulation complet doit être lu.

NOTE L'échec à lire le message complet peut se traduire par la perte de trace des frontières du message dans le train d'octets TCP.

4.3.1.6 Champ Session handle

Le champ Session Handle (Identification de session) doit être généré par la cible et retourné à l'émetteur en réponse à une demande RegisterSession. L'émetteur doit l'insérer dans toutes les demandes de commande d'encapsulation ultérieures (envoyées à l'aide des commandes énumérées dans le Tableau 184) exigeant des sessions sur cette cible particulière. Si c'est la cible qui déclenche et envoie une commande à l'émetteur, la cible doit inclure ce champ dans la demande qu'elle envoie à l'émetteur.

NOTE Certaines commandes (par exemple, Nop) n'exigent pas d'identification de session, même si une session a été établie. La spécification d'une commande particulière indiquera si une session n'est pas exigée.

4.3.1.7 Champ Status

La valeur dans le champ Status (statut) indique si, oui ou non, le destinataire était capable d'exécuter la commande d'encapsulation demandée. Une valeur de zéro dans une réponse doit indiquer une exécution réussie de la commande. Dans toutes les demandes émises par l'expéditeur, le champ Status doit contenir zéro. Si le destinataire reçoit une demande avec un champ Status différent de zéro, la demande doit être ignorée et aucune réponse ne doit être générée.

NOTE Ce champ ne reflète pas les erreurs qui sont générées par un paquet de protocole encapsulé contenu dans la partie "données" du message. Par exemple, une erreur rencontrée au cours du traitement d'un nœud final d'un service d'établissement d'attributs (Set Attributes) serait retournée via le mécanisme d'erreur spécifié de Type 2.

Les codes de statut doivent être tels que montrés dans le Tableau 185.

Tableau 185 – Codes de statut d'encapsulation

Code de statut	Description
0x00	Success (Succès)
0x01	L'expéditeur a émis une commande d'encapsulation qui n'est pas valide ou n'est pas prise en charge.

Code de statut	Description
0x02	Ressources mémoire insuffisantes dans le récepteur pour gérer la commande. Ceci n'est pas une erreur d'application. Cette erreur se produit uniquement lorsque la couche d'encapsulation ne peut obtenir les ressources mémoire dont elle a besoin.
0x03	Données formées médiocrement ou incorrectes dans la partie "données" du message d'encapsulation.
0x04 à 0x63	Réservés pour usage hérité[a]
0x64	Un émetteur a utilisé une identification de session invalide pour envoyer un message d'encapsulation à la cible.
0x65	La cible a reçu un message dont la longueur n'est pas valide (voir 4.3.1.5).
0x66 à 0x68	Réservés pour usage hérité[a]
0x69	Version de protocole non prise en charge
0x6A à 0xFFFF	Réservés pour des extensions futures[b]

[a] La mention "Réservé pour usage hérité" désigne des codes de statut qui ont été définis avant la publication de la présente norme. Leur usage n'est pas défini dans la présente norme. Les appareils ne doivent pas utiliser ces codes de statut s'il n'a pas été pris connaissance de leur usage hérité.

[b] Les codes de statut portant la mention "Réservé pour des extensions futures" ne doivent pas être utilisés.

4.3.1.8 Champ Sender context

L'expéditeur de la commande doit attribuer la valeur dans le champ Sender Context (contexte d'expéditeur) de l'en-tête. Le destinataire doit retourner cette valeur sans la modifier dans sa réponse. Les commandes n'impliquant pas de réponse peuvent ignorer ce champ.

L'expéditeur d'une demande de commande peut placer n'importe quelle valeur dans ce champ.

NOTE Ce champ peut être utilisé pour faire correspondre des demandes avec les réponses qui leur sont associées.

4.3.1.9 Champ Options

Ce champ a pour objet de fournir des bits qui modifient la signification des différentes commandes d'encapsulation. Les options et le comportement des options sont définis pour une commande donnée.

4.3.1.10 Champ Command specific data

La structure du champ de données spécifiques à une commande dépend du code de commande. Pour organiser leur champ de données spécifiques à une commande, la plupart des commandes utilisent les deux méthodes suivantes (séparément ou conjointement):

- utilisation d'une structure fixe;
- utilisation du format commun des paquets (spécifié en 4.3.3).

Le format commun des paquets permet aux commande de structurer leur champ de données spécifiques à une commande de manière extensible.

4.3.2 Descriptions des commandes

4.3.2.1 Nop

Un émetteur ou bien une cible peut envoyer une demande Nop. Aucune réponse ne doit être générée. La partie "données" de la demande doit avoir une longueur de 0 octet à 65 511

octets. Le destinataire doit ignorer toute donnée contenue dans le message. Une demande Nop n'exige pas l'établissement d'une session.

NOTE Une commande Nop fournit une façon pour un émetteur ou bien une cible de déterminer si la connexion TCP est encore ouverte.

L'en-tête d'encapsulation de la demande Nop doit être tel que montré dans le Tableau 186.

Tableau 186 – En-tête d'encapsulation de demande Nop

Nom de champ	Type de données	Valeur du champ
Command	UINT	Nop (0x00)
Length	UINT	Longueur des données spécifiques à une commande
Session handle	UDINT	N'importe quelle valeur (ignorée par la cible)
Status	UDINT	0 [a]
Sender context	ARRAY of 8 USHORT	Choisi par l'expéditeur
Options	UDINT	0

[a] Pour assurer la rétrocompatibilité, il ne faut pas définir d'option pour cette commande. Le destinataire doit rejeter les paquets dont le champ option n'est pas à zéro.

4.3.2.2 RegisterSession

Un émetteur doit envoyer une demande RegisterSession à une cible pour lancer une session. La commande RegisterSession n'exige pas l'établissement d'une session.

NOTE Voir 11.8 pour des informations détaillées sur l'établissement et le maintien d'une session.

L'en-tête d'encapsulation de la demande RegisterSession doit être tel que montré dans le Tableau 187.

Tableau 187 – En-tête d'encapsulation de demande RegisterSession

Nom de champ	Type de données	Valeur du champ
Command	UINT	RegisterSession (0x65)
Length	UINT	4
Session handle	UDINT	N'importe quelle valeur (ignorée par la cible)
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Toute valeur
Options	UDINT	0 [a]

[a] Pour assurer la rétrocompatibilité, il ne faut pas définir d'option pour cette commande. Le destinataire doit rejeter les paquets dont le champ option n'est pas à zéro.

Les paramètres dans la partie "données" doit déterminer la version du protocole et les éventuelles options de session comme montré dans le Tableau 188. La version de protocole doit être mise à 1. Aucun fanion d'option n'étant actuellement défini, les fanions d'options de session doivent être mis à 0.

NOTE Ce champ de fanions d'options n'est pas identique à celui que l'on trouve dans l'en-tête d'encapsulation.

Tableau 188 – Partie "données" de la demande RegisterSession

Champ	Type	Description
Protocol version	UINT	Version de protocole demandée (1)
Options flags	UINT	Options de session (0)

La cible doit envoyer une réponse RegisterSession pour indiquer qu'il a enregistré l'émetteur. La réponse doit avoir le même format que la demande, comme montré dans le Tableau 189.

Tableau 189 – En-tête d'encapsulation de réponse RegisterSession

Nom de champ	Type de données	Valeur du champ
Command	UINT	RegisterSession (0x65)
Length	UINT	4
Session handle	UDINT	Identification de session générée par la cible
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Valeur issue de la demande
Options	UDINT	0 [a]

[a] Pour assurer la rétrocompatibilité, il ne faut pas définir d'option pour cette commande. Le destinataire doit rejeter les paquets dont le champ option n'est pas à zéro.

Le champ Session Handle de l'en-tête doit contenir un identificateur généré par la cible que l'émetteur doit sauvegarder et insérer dans le champ Session Handle de l'en-tête de toutes les demandes ultérieures envoyées à la cible en question. Ce champ doit être valide seulement si le champ Status est zéro (0).

Le champ Sender Context de l'en-tête doit contenir les mêmes valeurs présentes dans la demande de l'expéditeur d'origine.

Si l'émetteur a été enregistré auprès de la cible, le champ Status doit être zéro (0). Si l'émetteur n'a pas été enregistré, le champ Status doit contenir le code de statut approprié, tel que défini ci-dessous:

- le code de statut 0x0001 doit être retourné si l'émetteur tente d'enregistrer plusieurs sessions actives sur la même connexion TCP;
- le code de statut 0x0002 doit être retourné si la cible ne dispose pas de suffisamment de ressources pour enregistrer l'émetteur;
- les autres codes de statut cités dans le Tableau 185 peuvent être utilisés pour signaler des erreurs générales d'encapsulation (par exemple, message d'encapsulation mal formé, longueur non valide);
- le code de statut 0x0069 doit être retourné en cas de discordance de Protocol Version ou d'Options, comme décrit ci-dessous.

La partie "données" de la réponse doit avoir le même format que la demande, comme montré dans le Tableau 190.

Tableau 190 – Partie "données" de la réponse RegisterSession

Champ	Type	Description
Protocol Version	UINT	Version de la demande RegisterSession si prise en charge. Si la version demandée n'est pas prise en charge, contient la version la plus élevée prise en charge.
Options	UINT	Fanions d'options de la demande RegisterSession si pris en charge. Si les fanions d'options demandés ne sont pas pris en charge, contient les fanions d'options pris en charge.

Le champ Protocol Version doit être égal à la version demandée si l'émetteur a été enregistré avec succès. Si la cible ne prend pas en charge la version demandée du protocole:

- la session ne doit pas être créée;
- le champ Status doit être défini sur "unsupported encapsulation protocol" (0x0069);
- la cible doit retourner la version prise en charge la plus élevée dans le champ Protocol Version.

Actuellement, aucun fanion d'option n'est défini. Afin de prendre en charge leur définition future, les cibles doivent contrôler la valeur des fanions d'options dans la demande RegisterSession. Si toutes les options demandées sont prises en charge, le champ Options de la réponse doit contenir la valeur demandée par l'émetteur. Si la cible ne prend pas en charge les options demandées:

- la session ne doit pas être créée;
- le champ Status doit être défini sur "unsupported encapsulation protocol" (0x0069);
- la cible doit retourner les options qu'elle prend en charge dans la réponse RegisterSession.

4.3.2.3 UnRegisterSession

Un émetteur ou bien une cible peut envoyer cette demande de mettre fin à la session. Le destinataire doit lancer une fermeture de la connexion TCP/IP sous-jacente lorsqu'il reçoit cette demande. La session doit aussi être arrêtée lorsqu'il est mis fin à la connexion de transport entre l'émetteur et la cible. Le destinataire doit accomplir toute mise au net associée exigée sur son côté. Il ne doit pas y avoir de réponse à cette commande, excepté en cas de réception de cette commande via UDP. Si la commande est reçue via UDP, le destinataire doit répondre avec le code de statut d'encapsulation 0x0001 (commande non valide ou non prise en charge).

Le format de la demande UnregisterSession doit être tel que montré dans le Tableau 191.

Tableau 191 – En-tête d'encapsulation de demande UnRegisterSession

Nom de champ	Type de données	Valeur du champ
Command	UINT	UnRegisterSession (0x65)
Length	UINT	4
Session Handle	UDINT	Identification de session provenant de RegisterSession
Status	UDINT	0
Sender Context	ARRAY of 8 USHORT	N'importe quelle valeur (ignorée par la cible)
Options	UDINT	0 [a]

[a] Pour assurer la rétrocompatibilité, il ne faut pas définir d'option pour cette commande. Le destinataire doit rejeter les paquets dont le champ option n'est pas à zéro.

Le destinataire ne doit pas rejeter UnRegisterSession en raison de la présence de valeurs inattendues dans l'en-tête d'encapsulation (identification de session non valide, statut différent de zéro, options différentes de zéro ou données de commande supplémentaires). Dans tous les cas, la connexion TCP doit être fermée.

NOTE Voir 11.8.3 pour obtenir plus de détails sur l'arrêt d'une session.

4.3.2.4 ListServices

La demande ListServices doit être envoyée pour déterminer quelles classes de services d'encapsulation l'appareil cible prend en charge. La commande ListServices n'exige pas l'établissement d'une session.

NOTE Chaque classe de services a un code de type unique et un nom ASCII facultatif.

L'en-tête d'encapsulation de la demande ListServices doit être tel que montré dans le Tableau 192.

Tableau 192 – En-tête d'encapsulation de demande ListServices

Nom de champ	Type de données	Valeur du champ
Command	UINT	ListServices (0x04)
Length	UINT	0
Session handle	UDINT	N'importe quelle valeur (ignorée par la cible)
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Choisi par l'expéditeur
Options	UDINT	0 [a]

[a] Pour assurer la rétrocompatibilité, il ne faut pas définir d'option pour cette commande. Le destinataire doit rejeter les paquets dont le champ option n'est pas à zéro.

Le destinataire doit répondre par un message d'encapsulation constitué de l'en-tête et de données, comme montré dans le Tableau 193 et dans le Tableau 194. La partie "données" de la réponse doit fournir des informations sur les services pris en charge.

Tableau 193 – En-tête d'encapsulation de réponse ListServices

Nom de champ	Type de données	Valeur du champ
Command	UINT	ListServices (0x04)
Length	UINT	Longueur des données spécifiques à une commande
Session handle	UDINT	Toute valeur (ignoré par le destinataire)
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Valeur issue de la demande
Options	UDINT	0 [a]

[a] Pour assurer la rétrocompatibilité, il ne faut pas définir d'option pour cette commande. Le destinataire doit rejeter les paquets dont le champ option n'est pas à zéro.

La partie "données" de la réponse doit contenir un compte d'éléments de 2 octets suivi d'une matrice d'éléments décrivant le(s) service(s) fourni(s), comme montré dans le Tableau 194.

Tableau 194 – Partie "données" de la réponse ListServices

Nom de champ	Type	Description
Item count	UINT	Nombre d'éléments à suivre
Target items	STRUCT of	Informations d'interface
	UINT	Item ID
	UINT	Item length
	UINT	Version du protocole encapsulé (doit être mis à 1)
	UINT	Capability Flags
	ARRAY of 16 USINT	Nom de service

L'ID d'élément doit identifier la classe de services. Une seule classe de services est définie, avec le code de type 0x100 et le nom “Communications”. Cette classe de service doit indiquer que le dispositif prend en charge l'encapsulation des PDU de Type 2. Tous les dispositifs qui prennent en charge l'encapsulation des PDU de Type 2 doivent prendre en charge la commande ListServices et la classe de services "Communications".

Le champ Version doit indiquer la version du service prise en charge par la cible pour aider à maintenir la compatibilité entre les applications.

Chaque service doit avoir un jeu différent de fanions de capacité. Les fanions réservés doivent être mis à zéro.

Les fanions de capacité, définis pour le service Communications, doivent être tels que montrés dans le Tableau 195.

Tableau 195 – Fanions de capacité de communications

Valeur du fanion	Description
Bits 0 à 4	Réservés pour usage hérité[a]
Bit 5	Si l'appareil prend en charge l'encapsulation des PDU de Type 2, ce bit doit être mis (= 1); autrement, il doit être effacé (= 0).
Bits 6 à 7	Réservés pour usage hérité[a]
Bit 8	Prend en charge les connexions basées sur UDP de classe de transport 0 ou 1
Bits 9 à 15	Réservés pour des extensions futures

[a] La mention "Réservé pour usage hérité" désigne des fanions qui ont été définis avant la publication de la présente norme. Leur usage n'est pas défini dans la présente norme. Les appareils ne doivent pas utiliser ces fanions s'il n'a pas été pris connaissance de leur usage hérité. Si un appareil reçoit un fanion réservé qu'il ne comprend pas, la réponse doit être traitée et le fanion ignoré.

Le champ Name doit permettre une chaîne ASCII terminée par le caractère NULL pouvant atteindre 16 octets au maximum à des fins de description uniquement. La limite de 16 octets doit inclure le caractère NULL.

4.3.2.5 ListIdentity

Un émetteur de connexion peut utiliser la demande ListIdentity pour localiser et identifier des cibles potentielles. Cette demande doit être envoyée comme un message de monodiffusion utilisant le protocole TCP ou le protocole UDP, ou comme un message de diffusion utilisant le

protocole UDP et ne doit pas exiger qu'une session soit établie. La réponse doit toujours être envoyée comme un message de monodiffusion.

Lorsqu'il reçoit un message de diffusion, l'appareil récepteur doit attendre pendant une période pseudo-aléatoire avant d'envoyer la réponse comme spécifié ci-dessous. Retarder l'envoi de la réponse favorise la propagation de toutes les demandes ARP et réponses ListIdentity résultantes issues des appareils cibles sur le réseau.

L'en-tête d'encapsulation de la demande ListIdentity doit être tel que montré dans le Tableau 196.

Tableau 196 – En-tête d'encapsulation de demande ListIdentity

Nom de champ	Type de données	Valeur du champ
Command	UINT	ListIdentity (0x63)
Length	UINT	0
Session handle	UDINT	N'importe quelle valeur (ignorée par la cible)
Status	UDINT	0
Sender context	UINT	MaxResponseDelay en ms (voir ci-dessous)
	ARRAY of 6 USHORT	Réservé; doit être ignoré par le destinataire; les valeurs doivent être 0
Options	UDINT	0 [a]

[a] Pour assurer la rétrocompatibilité, il ne faut pas définir d'option pour cette commande. Le destinataire doit rejeter les paquets dont le champ option n'est pas à zéro.

Un seul élément de réponse est défini pour cette commande, Target Identity (identité de cible), avec le code de type d'élément 0x0C. Cet élément doit être pris en charge (retourné) par tous les appareils de Type 2.

Un destinataire de la commande ListIdentity doit répondre par un message d'encapsulation constitué de l'en-tête et de données, comme montré dans le Tableau 197 et dans le Tableau 198. La partie "données" du message doit fournir des informations sur l'identité des cibles. La réponse doit être envoyée à l'adresse IP en provenance de laquelle la demande a été reçue.

Lorsqu'il reçoit la demande ListIdentity comme un message de diffusion UDP, le destinataire doit attendre avant d'envoyer la réponse. Lorsqu'il la reçoit comme un message de monodiffusion (via UDP ou TCP), il ne doit pas attendre.

Le délai imposé par le destinataire doit être une valeur aléatoire, en millisecondes, comprise entre 0 et l'intervalle MaxResponseDelay spécifié dans la demande ListIdentity. Si l'expéditeur spécifie une valeur de MaxResponseDelay de 0 ms, une valeur par défaut de 2 000 ms doit être utilisée par le destinataire. Si l'expéditeur spécifie une valeur de MaxResponseDelay comprise entre 1 ms et 500 ms, une valeur de 500 ms doit être utilisée par le destinataire. Une nouvelle valeur aléatoire doit être choisie pour chaque demande.

L'objectif de ce délai est de propager les réponses ListIdentity (et les messages ARP qui peuvent en résulter) pendant l'intervalle MaxResponseDelay. Il est donc important que chaque appareil génère une valeur de délai aléatoire unique. Les appareils doivent veiller à générer chacun une valeur unique, par exemple en amorçant leur générateur de nombres aléatoires avec une valeur unique telle que leur adresse IP ou leur adresse MAC Ethernet.

Il est possible que les appareils reçoivent des demandes ListIdentity de diffusion supplémentaires pendant le délai précédant la réponse, par exemple si plusieurs clients émettent la demande ListIdentity de diffusion. Il convient que les appareils puissent accepter et traiter simultanément au moins 2 demandes ListIdentity de diffusion en cours.

Une fois qu'il a émis une demande ListIdentity de diffusion, il convient qu'un client n'émette pas d'autre demande avant l'expiration de l'intervalle MaxResponseDelay associé à sa demande en cours. Le comportement du client en ce qui concerne la gestion des réponses ListIdentity reçues au-delà de l'intervalle MaxResponseDelay dépend du vendeur.

Tableau 197 – En-tête d'encapsulation de réponse ListIdentity

Nom de champ	Type de données	Valeur du champ
Command	UINT	List Identity (0x63)
Length	UINT	Longueur des données spécifiques à une commande
Session handle	UDINT	Toute valeur (ignoré par le destinataire)
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Valeur issue de la demande
Options	UDINT	0 [a]

[a] Pour assurer la rétrocompatibilité, il ne faut pas définir d'option pour cette commande. Le destinataire doit rejeter les paquets dont le champ option n'est pas à zéro.

La partie "données" de la réponse est structurée comme un Common Packet Format (format commun des paquets) qui contient un compte d'éléments de 2 octets suivi d'une matrice d'éléments fournissant l'identité de la cible, comme montré dans le Tableau 198.

Tableau 198 – Partie "données" de la réponse ListIdentity (succès)

Nom de champ	Type de données	Valeur du champ
Item count	UINT	Nombre d'éléments cibles à suivre
Target items	STRUCT of	Informations d'interface
	UINT	Item ID
	UINT	Item length
	ARRAY of octets	Données d'élément

L'élément Identity CFP 2 doit être retourné en premier; son format est tel que défini dans le Tableau 199. Une partie de cette définition d'élément suit la définition de réponse de service Get_Attribute_All de l'objet Identity (données retournées sur la base de l'instance une de cet objet). Contrairement à la plupart des champs dans le format commun des paquets, le champ Socket Address doit être envoyé dans un ordre big endian.

Actuellement, aucun élément supplémentaire n'est défini pour la réponse ListIdentity. Des éléments supplémentaires pourront être définis plus tard. Les destinataires de la réponse ListIdentity doivent ignorer les éléments inattendus.

Tableau 199 – Elément Identity CPF 2

Nom de champ	Type	Description
Item ID	UINT	ID d'élément d'Identity CPF 2 (0x0C)
Item length	UINT	Nombre d'octets dans l'élément qui suit (la longueur varie en fonction de la chaîne Product Name)
Version	UINT	Version du protocole d'encapsulation prise en charge (également retournée avec la réponse Register Session)
Socket Address	STRUCT of:	Adresse du port de connexion (voir 4.3.3.3.3)
	INT	sin_family (big-endian)
	UINT	sin_port (big-endian)
	UDINT	sin_addr (big-endian)
	ARRAY of USINT	sin_zero (length of 8) (big-endian)
Vendor ID[a]	UINT	ID de vendeur des fabricants d'appareils
Device Type[a]	UINT	Type d'appareil du produit
Product Code[a]	UINT	Code de produit attribué par rapport au type d'appareil
Revision[a]	USINT[2]	Révision de l'appareil
Status[a]	WORD	Statut actuel de l'appareil
Serial Number[a]	UDINT	Numéro de série de l'appareil
Product Name[a]	SHORT_STRING	Description de l'appareil lisible par l'homme
State[b]	USINT	Etat actuel de l'appareil

[a] Ces paramètres sont définis en plus par l'attribut d'instance correspondant de l'objet Identity.

[b] L'attribut State est un attribut facultatif de l'objet Identity. S'il n'est pas mis en œuvre, sa valeur doit être 0xFF.

4.3.2.6 ListInterfaces

La demande facultative ListInterfaces doit être utilisée par un émetteur de connexion pour identifier des interfaces de communications autres que de Type 2 associées à la cible. Une session peut ne pas être établie pour envoyer cette commande.

L'en-tête d'encapsulation de la demande ListInterfaces doit être tel que montré dans le Tableau 200.

Tableau 200 – En-tête d'encapsulation de demande ListInterfaces

Nom de champ	Type de données	Valeur du champ
Command	UINT	List Interfaces (0x64)
Length	UINT	0
Session handle	UDINT	N'importe quelle valeur (ignorée par la cible)
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Choisi par l'expéditeur
Options	UDINT	0 [a]

[a] Pour assurer la rétrocompatibilité, il ne faut pas définir d'option pour cette commande. Le destinataire doit rejeter les paquets dont le champ option n'est pas à zéro.

Si elle est prise en charge, le destinataire d'une commande de demande ListInterfaces doit répondre par un message d'encapsulation constitué de l'en-tête et de données, comme montré dans le Tableau 201.

Tableau 201 – En-tête d'encapsulation de réponse ListInterfaces

Nom de champ	Type de données	Valeur du champ
Command	UINT	List Interfaces (0x64)
Length	UINT	Longueur des données spécifiques à une commande
Session handle	UDINT	Toute valeur (ignoré par le destinataire)
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Valeur issue de la demande
Options	UDINT	0 [a]

[a] Pour assurer la rétrocompatibilité, il ne faut pas définir d'option pour cette commande. Le destinataire doit rejeter les paquets dont le champ option n'est pas à zéro.

La partie "données" de la réponse contient une matrice d'éléments fournissant des informations d'interface. Elle est structurée comme un Common Packet Format (format commun des paquets) qui contient un compte d'éléments de 2 octets suivi d'une matrice d'éléments

Actuellement, aucun élément public n'est défini pour la réponse ListInterfaces. Si aucun élément n'est inclus, Item Count doit être mis à 0.

Certains appareils hérités peuvent retourner le message "Réservé pour les éléments à usage hérité" (voir 4.3.3). Ces éléments doivent être ignorés à moins que l'appareil récepteur n'ait une connaissance explicite du format et de l'usage hérités.

4.3.2.7 SendRRData

Une demande SendRRData doit transférer un paquet encapsulé de demande/réponse entre l'émetteur et la cible, l'émetteur lançant la commande. Les paquets réels de demande/réponse doivent être encapsulés dans la partie "données" du message et doivent être de la responsabilité de la cible et de l'émetteur.

NOTE Lorsqu'elles sont utilisées pour encapsuler des PDU de Type 2, la demande et la réponse SendRRData sont utilisées pour envoyer des messages d'UCMM encapsulés (voir Article 11).

L'en-tête d'encapsulation de la demande SendRRData doit être tel que montré dans le Tableau 202.

Tableau 202 – En-tête d'encapsulation de demande SendRRData

Nom de champ	Type de données	Valeur du champ
Command	UINT	SendRRData (0x6F)
Length	UINT	Longueur des données spécifiques à une commande
Session handle	UDINT	Session handle
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Choisi par l'expéditeur
Options	UDINT	0 [a]

[a] Pour assurer la rétrocompatibilité, il ne faut pas définir d'option pour cette commande. Le destinataire doit rejeter les paquets dont le champ option n'est pas à zéro.

La partie "données" du message doit contenir les paramètres de demande et le paquet de protocole encapsulé, comme montré dans le Tableau 203.

Tableau 203 – Partie "données" de la demande SendRRData

Nom de champ	Type	Description
Interface handle	UDINT	Shall be 0
Timeout	UINT	Temporisation d'opération (0 à 65 535)
Encapsulated protocol packet	ARRAY of octets	Voir la spécification de Common Packet Format en 4.3.3

Le champ Interface handle doit identifier l'interface de communications à laquelle la demande est dirigée. Cette identification doit être 0 pour les PDU d'encapsulation de Type 2.

La cible doit abandonner l'opération demandée après que la temporisation expire. Lorsque le champ Timeout se situe dans la plage 1 à 65 535, la temporisation doit être définie sur ce nombre de secondes. Lorsque le champ Timeout est mis à 0, le protocole d'encapsulation ne doit pas avoir sa propre temporisation. Il doit s'appuyer sur le mécanisme de temporisation du protocole encapsulé.

Lorsque la commande SendRRData est utilisée pour encapsuler des PDU de Type 2, le champ Timeout doit être mis à 0 et doit être ignoré par la cible.

Le paquet de protocole encapsulé doit être codé dans un Common Packet Format, comme montré en 4.3.3.

La réponse SendRRData, telle que montrée dans le Tableau 204, doit contenir des données en réponse à la demande SendRRData. La réponse à la demande de protocole encapsulé d'origine doit être contenue dans la partie "données" de la réponse SendRRData.

Tableau 204 – En-tête d'encapsulation de réponse SendRRData

Nom de champ	Type de données	Valeur du champ
Command	UINT	SendRRData (0x6F)
Length	UINT	Longueur des données spécifiques à une commande
Session handle	UDINT	Valeur issue de la demande
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Valeur issue de la demande
Options	UDINT	0 [a]

[a] Pour assurer la rétrocompatibilité, il ne faut pas définir d'option pour cette commande. Le destinataire doit rejeter les paquets dont le champ option n'est pas à zéro.

Le format de la partie "données" du message de réponse doit être le même que celui du message de demande SendRRData. La demande et la réponse utilisant un format commun, le message de réponse contient un champ Timeout; cependant, il n'est pas utilisé.

4.3.2.8 SendUnitData;

La demande SendUnitData doit envoyer des messages connectés encapsulés. Une réponse peut ne pas être retournée. Cette commande peut être utilisée lorsque le protocole encapsulé a son propre mécanisme sous-jacent de transport de bout en bout. La demande SendUnitData peut être envoyée depuis l'une ou l'autre des extrémités de la connexion TCP.

NOTE Lorsqu'elle est utilisée pour encapsuler des PDU de Type 2, la commande SendUnitData est utilisée pour envoyer des données connectées de Type 2 à la fois dans le sens O⇒T et dans le sens T⇒O.

Le format de la demande SendUnitData doit être tel que montré dans le Tableau 205.

Tableau 205 – En-tête d'encapsulation de demande SendUnitData

Nom de champ	Type de données	Valeur du champ
Command	UINT	SendUnitData (0x70)
Length	UINT	Longueur de la partie "données"
Session handle	UDINT	Session handle
Status	UDINT	0
Sender context	ARRAY of 8 USHORT	Toute valeur (ignoré par la cible)
Options	UDINT	0 [a]

[a] Pour assurer la rétrocompatibilité, il ne faut pas définir d'option pour cette commande. Le destinataire doit rejeter les paquets dont le champ option n'est pas à zéro.

La partie "données" du message doit contenir les paramètres de demande ainsi que le paquet de protocole encapsulé, comme montré dans le Tableau 206.

Tableau 206 – Partie "données" de la demande SendUnitData

Nom de champ	Type	Description
Interface handle	UDINT	Shall be 0
Timeout	UINT	Temporisation d'opération (doit être 0)
Encapsulated protocol packet	ARRAY of octets	Voir la spécification de Common Packet Format en 4.3.3

Les champs Interface handle et Timeout doivent être mis à zéro. Le champ Timeout n'est pas utilisé car aucune réponse n'est générée à réception d'une commande SendUnitData.

4.3.3 Format commun des paquets

4.3.3.1 Généralités

Le format commun des paquets (CPF) définit un format normalisé pour les paquets de protocole qui sont transportés avec le protocole d'encapsulation. Le format commun des paquets est un mécanisme d'usage général conçu pour prendre en charge les futurs types de paquet ou d'adresse.

Le format commun des paquets doit être constitué d'un compte d'éléments, suivi d'un nombre d'éléments (voir Tableau 207). Certains éléments sont classés comme étant des "éléments adresse" (transportant des informations d'adressage) ou des "éléments données" (transportant des données encapsulées). Le nombre d'éléments à inclure dépend de la commande d'encapsulation et de l'usage de cette commande. Le Paragraphe 4.3.3.4 spécifie les éléments de format commun des paquets valides pour les différentes commandes et les différents usages.

Tableau 207 – Format commun des paquets

Nom de champ	Type de données	Description
Item count	UINT	Nombre d'éléments à suivre
Item #1	Item Struct (voir ci-dessous)	Premier élément CPF
Item #2	Item Struct (voir ci-dessous)	Deuxième élément CPF
...	...	...
Item #n	Item Struct (voir ci-dessous)	n^e élément CPF

La structure des éléments adresse et donnée doit être telle que montrée dans le Tableau 208.

Tableau 208 – Format des éléments CPF

Nom de champ	Type	Description
Type ID	UINT	Type de l'élément encapsulé
Length	UINT	Longueur en octets des données à suivre
Data	Variable	Les données (si longueur > 0)

Les numéros de Type ID des éléments doivent être tels que montrés dans le Tableau 209.

Tableau 209 – Numéros de Type ID des éléments

Numéro d'identification d'élément	Type d'élément	Description
0x0000	adresse	Null (utilisé pour les messages d'UCMM). Indique que l'acheminement d'encapsulation n'est PAS nécessaire. La cible est locale (Ethernet) ou les informations d'acheminement se trouvent dans un élément données.
0x0001 à 0x000B		Réservés pour usage hérité[a]

Numéro d'identification d'élément	Type d'élément	Description
0x000C		Réponse ListIdentity
0x000D à 0x0085		Réservés pour usage hérité[a]
0x0086 à 0x0090		Réservés pour des extensions futures[b]
0x0091		Réservés pour usage hérité[a]
0x0092 à 0x00A0		Réservés pour des extensions futures[b]
0xA1	adresse	Basé sur la connexion (utilisé pour les messages connectés)
0x00A2 à 0x00A4		Réservés pour usage hérité[a]
0x00A5 à 0x00B0		Réservés pour des extensions futures[b]
0x00B1	données	Paquet de transport connecté
0x00B2	données	Message non connecté
0x00B3 à 0x00FF		Réservés pour des extensions futures[b]
0x0100		Réponse ListServices
0x0101 à 0x010F		Réservés pour usage hérité[a]
0x0110 à 0x7FFF		Réservés pour des extensions futures[b]
0x8000	données	Sockaddr Info, émetteur vers cible
0x8001	données	Sockaddr Info, cible vers émetteur
0x8002		Elément adresse "sequenced"
0x8003 à 0xFFFF		Réservés pour des extensions futures[b]

[a] La mention "Réservé pour usage hérité" désigne des ID d'élément qui ont été définis avant la publication de la présente norme. Leur comportement n'est pas défini dans la présente norme. Les appareils ne doivent pas utiliser ces ID d'élément s'il n'a pas été pris connaissance de leur usage hérité.

[b] Les produits conformes à cette spécification ne doivent pas utiliser de codes de commande situés dans cette plage.

4.3.3.2 Eléments adresse

4.3.3.2.1 Null

L'élément adresse "null" doit contenir seulement l'identificateur de type et la longueur, comme montré dans le Tableau 210. La longueur doit être zéro. Aucune donnée ne doit suivre la longueur. Comme l'élément adresse "null" ne contient aucune information d'acheminement, il doit être utilisé lorsque le paquet de protocole lui-même contient toute information d'acheminement nécessaire. L'élément adresse "null" doit être utilisé pour les messages non connectés (Unconnected Messages).

Tableau 210 – Elément adresse "null"

Nom de champ	Type de données	Valeur du champ
Type ID	UINT	0
Length	UINT	0

4.3.3.2.2 Connected

Cet élément adresse doit être utilisé lorsque le protocole encapsulé est orienté connexion. Les données doivent contenir un identificateur de connexion, montré dans le Tableau 211.

NOTE Les identificateurs de connexion sont échangés dans le service Forward_Open service du Connection Manager.

Tableau 211 – Elément adresse "connected"

Nom de champ	Type de données	Valeur du champ
Type ID	UINT	0xA1
Length	UINT	4
Data	UDINT	Identificateur de connexion

4.3.3.2.3 Elément adresse "sequenced"

Cet élément adresse doit être utilisé pour les données connectées de classe 0 et de classe 1. Les données doivent contenir un identificateur de connexion et un numéro de séquence, comme montré dans le Tableau 212. L'utilisation est décrite avec plus de détails en 11.3.3.

Tableau 212 – Elément adresse "sequenced"

Nom de champ	Type	Valeur du champ
Type ID	UINT	0x8002
Length	UINT	8
Data	UDINT	Identificateur de connexion
	UDINT	Numéro de séquence

4.3.3.3 Eléments données

4.3.3.3.1 Unconnected

L'élément données qui encapsule un message non connecté doit être tel que montré dans le Tableau 213.

Tableau 213 – Elément données "unconnected"

Nom de champ	Type	Valeur du champ
Type ID	UINT	0xB2
Length	UINT	Longueur, en octets, du message non connecté
Data	Variable	Le message non connecté

Le format du champ Data dépend du protocole encapsulé. Lorsqu'il est utilisé pour encapsuler des PDU de Type 2, le format du champ Data doit être identique à celui de la PDU destinée au Message Router. L'en-tête UCMM, défini en 4.1.3, ne doit pas être présent. Le champ context dans l'en-tête d'encapsulation doit être utilisé pour la correspondance demande/réponse non connectée.

4.3.3.3.2 Connected

L'élément données qui encapsule une PDU de transport connectée doit être tel que montré dans le Tableau 214.

Tableau 214 – Elément données "connected"

Nom de champ	Type	Valeur du champ
Type ID	UINT	0xB1
Length	UINT	Longueur, en octets, de la PDU de transport
Data	Variable	La PDU de transport

Le format du champ Data dépend du protocole encapsulé. Lorsqu'il est utilisé pour encapsuler des PDU de Type 2, le format du champ Data doit être identique à celui de la PDU de transport connectée.

4.3.3.3.3 Sockaddr Info

Les éléments Sockaddr Info doivent être utilisés pour communiquer les informations d'adresse IP ou de port nécessaires à la création de connexions de classe 0 ou de classe 1. Ce sont des éléments séparés pour les informations de port de connexion émetteur vers cible et cible vers émetteur. Ces éléments sont présents sous la forme de données supplémentaires dans les services de demande et de réponse Forward_Open/Large_Forward_Open encapsulés dans un message SendRRData. Le Paragraphe 11.2.4 spécifie l'usage de ces éléments dans le contexte de la création de connexions CIP.

Les éléments Sockaddr Info doivent avoir la structure montrée dans le Tableau 215.

Tableau 215 – Eléments Sockaddr Info

Nom de champ	Type	Valeur du champ
Type ID	UINT	0x8000 pour O⇒T, 0x8001 pour T⇒O
Length	UINT	16 (octets)
sin_family	INT	Doit être AF_INET = 2. Ce champ doit être envoyé dans un ordre big endian.
sin_port	UINT	Pour les connexions point à point, sin_port doit être défini sur le port UDP auquel les paquets associés à cette connexion de Type 2 seront envoyés. Pour les connexions point à point, il est recommandé d'utiliser le port UDP enregistré (0x8AE). Lorsqu'il est utilisé avec une connexion de multidiffusion, le champ sin_port doit être défini sur le numéro de port UDP enregistré (0x08AE) et traité par le destinataire comme "sans influence". Ce champ doit être envoyé dans un ordre big endian.
sin_addr	UDINT	Pour les connexions de multidiffusion, sin_addr doit être défini sur l'adresse de multidiffusion IP à laquelle les paquets associés à cette connexion de Type 2 seront envoyés. Lorsqu'il est utilisé avec une connexion point à point, le champ sin_addr doit être traité par le destinataire comme "sans influence". Il est recommandé que l'expéditeur mette sin_addr à 0 pour les connexions point à point. Ce champ doit être envoyé dans un ordre big endian.
sin_zero	ARRAY of 8 USINT	Valeur zéro recommandée; non appliqué

Le format de l'élément Sockaddr Info a été créé d'après la structure sockaddr_in définie dans la Spécification Winsock, version 1.1.

4.3.3.4 Résumé des usages valides des éléments de format commun des paquets

Le format commun des paquets est utilisé avec les commandes d'encapsulation suivantes:

- réponse ListIdentity;
- réponse ListInterfaces;
- réponse ListServices;
- demande et réponse SendRRData;
- SendUnitData;
- paquets de classes de transport 0 et 1 (pas d'en-tête d'encapsulation).

Le Tableau 216 montre l'usage valide des éléments CPF pour les différentes commandes d'encapsulation.

NOTE L'Article 11 présente en détail les formats et usages associés aux messages connectés et non connectés de Type 2.

Tableau 216 – Usage des éléments CPF

Command	Eléments CPF requis	Eléments CPF facultatifs	Action en cas de présence d'éléments CPF inattendus ou non définis
réponse ListIdentity;	Elément de réponse ListIdentity	Aucun	Ignorer les éléments
réponse ListInterfaces;	Aucun	Les appareils hérités peuvent retourner des éléments "Réservés pour usage hérité"	Ignorer les éléments
réponse ListServices;	Elément ListServices du service Communications	Les appareils hérités peuvent retourner des éléments "Réservés pour usage hérité"	Ignorer les éléments
Demande SendRRData	Elément adresse suivi de l'élément données	Lorsque cette demande est utilisée pour encapsuler le service Forward_Open, des éléments Sockaddr_info supplémentaires peuvent être présents, comme spécifié à l'Article 11	Retourner l'erreur (0x0003)
Réponse SendRRData	Elément adresse suivi de l'élément données	Lorsque cette réponse est utilisée pour encapsuler la réponse Forward_Open, des éléments Sockaddr_info supplémentaires peuvent être présents, comme spécifié à l'Article 11	Signaler l'erreur à l'application appelante. Pour la réponse Forward_Open, la connexion ne doit pas être établie au niveau de l'émetteur
SendUnitData;	Elément adresse suivi de l'élément données	Aucun	Rejeter le message
Paquet de classe 0/1	Elément adresse suivi de l'élément données	Aucun	Rejeter le message

5 Syntaxe de transfert

5.1 Codage compact

5.1.1 Règles de codage

Le Paragraphe 5.1 décrit le moyen par lequel les types de données définis dans la présente norme doivent être codés/transférés sur le réseau de commande. La définition de syntaxe

abstraite accompagnée d'un jeu particulier de règles de codage doit aboutir à la syntaxe de transfert. Pour les données d'utilisateur d'application, un seul jeu de règles de codage doit être défini ("Compact Encoding" Codage compact), aboutissant à la syntaxe de transfert compacte.

Les règles de codage compact doivent commencer par les règles de codage définies dans l'ISO/CEI 8825-1. Le codage compact applique alors des règles d'optimisation, en commençant par l'unité de données de service (Service Data Unit, SDU) la plus extérieure et en progressant vers chaque SDU encapsulée successive. Le codage compact doit définir un mécanisme de codage plus efficace en réduisant la quantité des informations (surdébit) transférées entre les appareils.

La différence entre une valeur codée par codage compact et une valeur codée par codage ASN.1 doit être la suppression des champs décrivant le type et la longueur des informations. Les composants TAG et LENGTH d'une valeur codée par codage ASN.1 ne doivent pas être émis sur le réseau de commande. En outre, les règles du codage compact doivent indiquer que les règles d'ordre de classement des octets sont l'inverse de celles vues dans l'ASN.1.

Considérant les conditions énumérées en 5.1.2, des règles générales doivent être appliquées à une valeur codée par codage ASN.1 pour générer une valeur codée par codage compact. Les règles générales doivent être comme suit:

- supprimer les octets d'identificateur ("Identifier");
- supprimer les octets “TAG” spécifiés par ASN.1;
- supprimer les octets de longueur ("Length");
- supprimer les octets “LENGTH” spécifiés par ASN.1;
- inverser l'ordre des octets pour les octets de contenu multiples.

5.1.2 Contraintes de codage

La représentation d'une valeur variable utilisant le codage compact doit être possible avec les restrictions suivantes:

- le type de la variable doit être de longueur fixe et ne doit avoir aucun champ conditionnel ou facultatif;
- le codage d'une valeur donnée doit être représenté avec un nombre constant d'octets dérivés de la spécification de type de cette variable.

5.1.3 Exemples

5.1.3.1 Codage de BOOL

Le codage de BOOLEAN doit être accompli sur un seul OCTET comme décrit dans le Tableau 217.

Tableau 217 – Codage BOOLEAN

Si la valeur est:	Alors:
FALSE	le bit 0 de l'octet est 0 (’00’H)
TRUE	le bit 0 de l'octet est 1 (’01’H)

Un BOOL FALSE doit être représenté comme montré dans le Tableau 218.

Tableau 218 – Exemple de codage compact d'une valeur BOOL

Numéro d'octet	1er
BOOL	00

5.1.3.2 Codage de SignedInteger

Le codage de SignedInteger doit être accompli comme décrit dans le Tableau 219.

Tableau 219 – Codage des valeurs SignedInteger

Numéro d'octet	1er	2e	3e	4e	5e	6e	7e	8e
SINT	0LSB							
INT	0LSB	1LSB						
DINT	0LSB	1LSB	2LSB	3LSB				
LINT	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB
NOTE L'exemple du Tableau 220 illustre le codage d'une variable de type DINT dont la valeur est 0x12345678.								

Tableau 220 – Exemple de codage compact d'une valeur SignedInteger

Numéro d'octet	1er	2e	3e	4e
DINT	78	56	34	12

5.1.3.3 Codage de UnsignedInteger

Le codage de UnsignedInteger doit être accompli comme décrit dans le Tableau 221.

Tableau 221 – Valeurs UnsignedInteger

Numéro d'octet	1er	2e	3e	4e	5e	6e	7e	8e
USINT	0LSB							
UINT	0LSB	1LSB						
UDINT	0LSB	1LSB	2LSB	3LSB				
ULINT	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB
NOTE Le Tableau 222 illustre le codage d'une variable de type UDINT dont la valeur est 0xAABBCCDD.								

Tableau 222 – Exemple de codage compact d'un UnsignedInteger

Numéro d'octet	1er	2e	3e	4e
UDINT	DD	CC	BB	AA

5.1.3.4 Codage de FixedLengthReal

Le codage de FixedLengthReal doit être accompli comme décrit dans le Tableau 223.

Tableau 223 – Valeurs FixedLengthReal

Numéro d'octet	1er	2e	3e	4e	5e	6e	7e	8e
REAL	0LSB	1LSB	2LSB	3LSB				
LREAL	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

Le Tableau 224 illustre le codage d'une variable (Float1) dont le type est REAL et dont la valeur est Float1: = 10,0.

NOTE 1 L'affectation de la valeur utilise la notation CEI 61131-3. La valeur ASN.1 est {'41200000'H} dans le format de l'IEEE: 1,25*23, l'exposant est 130 (biais 127), la fraction est 25.

Tableau 224 – Exemple de codage compact d'une valeur REAL

Contenus d'octet	0LSB	1LSB	2LSB	3LSB
REAL	00	00	20	41

Le Tableau 225 illustre le codage d'une variable (Float2) dont le type est LREAL et dont la valeur est Float2: = –100,0.

NOTE 2 La valeur ASN.1 est {'C059000000000000'H} dans le format de l'IEEE: 1,562 5*26, l'exposant est 1 029 (biais 1 023), la fraction est 0,562 5.

Tableau 225 – Exemple de codage compact d'une valeur LREAL

Contenus d'octet	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB
LREAL	00	00	00	00	00	00	59	C0

5.1.3.5 Codages de temps

Le Tableau 226 illustre le codage de nombres réels (REAL) à longueurs fixes (fixedlengths).

Tableau 226 – Valeurs FixedLengthReal

Numéro d'octet	1er	2e	3e	4e	5e	6e	7e	8e
TIME	0LSB	1LSB	2LSB	3LSB				
DATE	0LSB	1LSB						
TIME_OF_DAY	0LSB	1LSB	2LSB	3LSB				
DATE_AND_TIME	0LSB Temps	1LSB Temps	2LSB Temps	3LSB Temps	0LSB Date	1LSB Date		
FTIME	0LSB	1LSB	2LSB	3LSB				
LTIME	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

5.1.3.6 Codages de chaîne

Le Paragraphe 5.1.3.6 donne des exemples de codage compact appliqué à des valeurs de données STRING, STRING2, STRINGN et SHORT_STRING.

NOTE Le type de chaîne préférentiel pour les données de chaîne fournies par l'utilisateur est STRING2 en raison des exigences internationales relatives aux chaînes de caractères.

Le Tableau 227 donne une illustration générique du codage d'une valeur STRING.

Tableau 227 – Valeur STRING

	Contenu (charcount)		Contenu (contenu de chaîne)
STRING	0LSB	1LSB	0LSB
			Caractères de 1 octet

Le Tableau 228 donne une illustration générique du codage d'une valeur STRING2.

Tableau 228 – Valeur STRING2

	Contenu (charcount)		Contenu (string2contents)	
STRING2	0LSB	1LSB	0LSB	1LSB
				Caractères de 2 octet

Le Tableau 229 donne une illustration générique du codage d'une valeur STRINGN.

Tableau 229 – Valeur STRINGN

	Contenu (charsize)		Contenu (charcount)		Contenu (stringNcontents)	
STRINGN	0LSB	1LSB	0LSB	1LSB	0LSB	NLSB
						Caractères de N octets

Le Tableau 230 donne une illustration générique du codage d'une valeur SHORT_STRING.

Tableau 230 – Valeur SHORT_STRING

	Contenu (charcount)	Contenu (short_string)
SHORT_STRING	0LSB	0LSB
		Caractères de 1 octet

Le Tableau 231 illustre le codage d'une variable chaîne dont le contenu est "Mill". Des exemples de codage de tous les types de chaîne sont présentés. Le codage de caractère est spécifié dans l'ISO/CEI 10646. L'équivalent hexadécimal est: {'4D696C6C'H} pour le codage 8 bits.

Le Tableau 231 code "Mill" comme un type STRING.

Tableau 231 – Exemple de codage compact d'une valeur STRING

	Contenu (charcount)		Contenu (contenu de chaîne)			
STRING	04	00	4D	69	6C	6C

Le Tableau 232 code "Mill" comme un type STRING2.

Tableau 232 – Exemple de codage compact d'une valeur STRING2

	Contenu (charcount)		Contenu (contenu string2)							
STRING2	04	00	4D	00	69	00	6C	00	6C	00

Le Tableau 233 code "Mill" comme un type SHORT_STRING.

Tableau 233 – Type SHORT_STRING

	Contenu (charcount)	Contenu (contenu short_string)			
SHORT_STRING	04	4D	69	6C	6C

5.1.3.7 Codage de FixedLengthBitString

Le Paragraphe 5.1.3.7 donne des exemples de codage compact appliqué à des valeurs de données SWORD, WORD, DWORD, LWORD. La Figure 16 illustre les règles de placement des bits associées au codage compact d'un FixedLengthBitString.

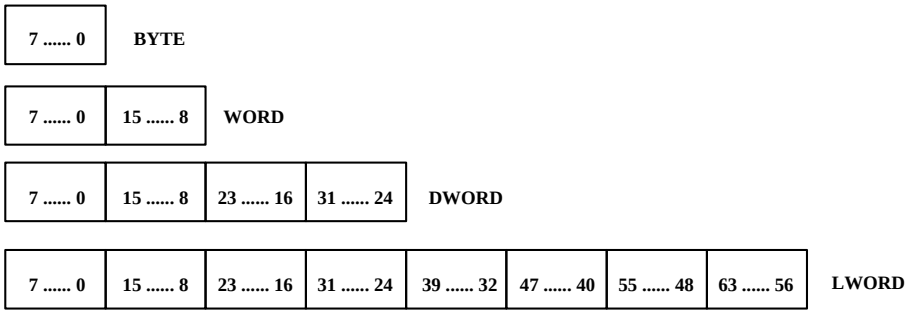
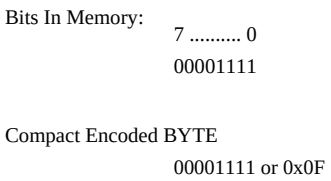


Figure 16 – Règles de placement de bits associées au codage compact de FixedLengthBitString

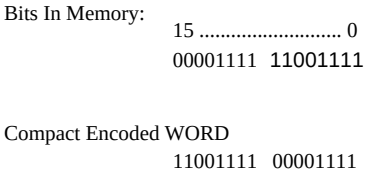
La Figure 17 à la Figure 20 illustrent le codage de SWORD, WORD, DWORD et LWORD.



Légende

Anglais	Français
Bits In Memory:	Bits en mémoire:
Compact Encoded BYTE	BYTE à codage compact

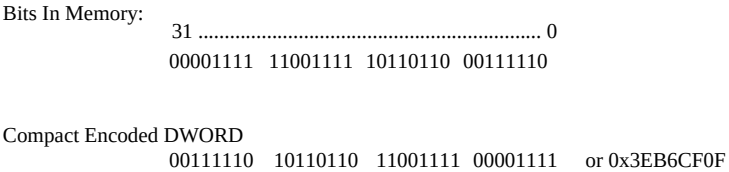
Figure 17 – Exemple de codage compact d'un FixedLengthBitString SWORD



Légende

Anglais	Français
Bits In Memory:	Bits en mémoire:
Compact Encoded WORD	WORD à codage compact

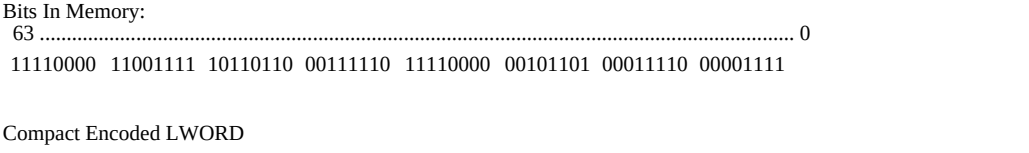
Figure 18 – Exemple de codage compact d'un FixedLengthBitString WORD



Légende

Anglais	Français
Bits In Memory:	Bits en mémoire:
Compact Encoded DWORD	DWORD à codage compact

Figure 19 – Exemple de codage compact d'un FixedLengthBitString DWORD



Légende

Anglais	Français
Bits In Memory:	Bits en mémoire:
Compact Encoded LWORD	LWORD à codage compact

Figure 20 – Exemple de codage compact d'un FixedLengthBitString LWORD

5.1.3.8 Codage de matrice

5.1.3.8.1 Codage de matrice unidimensionnelle

Le codage de Array doit utiliser les règles de codage pour les types de données pour chaque élément et concaténer les éléments qui composent la matrice. Les valeurs codées des éléments de la matrice doivent être codées dans le même ordre qu'elles sont déclarées dans la spécification de type ou de variable correspondante de l'ASN.1. Ces éléments peuvent être de n'importe quel type de données.

La définition au style ASN.1 d'une matrice unidimensionnelle dans le réseau de commande doit être:

```
ARRAY::= SEQUENCE OF { array_dimension_low_bound,
                       array_dimension_high_bound, NetworkData }
```

```
Assume the following array definition::
ARRAY1::= SEQUENCE OF {array_dimension_low_bound:= 0,
          array_dimension_high_bound:= 1, UINT}
```

L'insertion des valeurs UINT {1,2} dans la définition de la matrice donne le codage spécifié dans le Tableau 234.

Tableau 234 – Exemple de codage compact d'un ARRAY à une seule dimension

Numéro d'octet	1er	2e	3e	4e
ARRAY	01	00	02	00

5.1.3.8.2 Codage de matrice bidimensionnelle

```
ARRAY::= SEQUENCE OF { array_dimension_low_bound,
array_dimension_high_bound,
     SEQUENCE OF { array_dimension_low_bound,
array_dimension_high_bound, NetworkData } }
```

5.1.3.8.3 Codage de matrice tridimensionnelle

```
ARRAY::= SEQUENCE OF { array_dimension_low_bound,
   array_dimension_high_bound,
   SEQUENCE OF { array_dimension_low_bound,
   array_dimension_high_bound,
   SEQUENCE OF { array_dimension_low_bound,
    array_dimension_high_bound, NetworkData } } }
```

Sachant que les données de réseau de commande peuvent comprendre des ElementaryData ou des DerivedData, une nouvelle spécification de type ou de variable peut être requise avant d'émettre les valeurs pour l'ARRAY.

Une matrice multidimensionnelle doit être vue sur la ligne comme une matrice à une seule dimension. L'ordre des éléments de la matrice doit être conservé par le biais d'une séquence de condensation/décompression, suivie des nœuds d'extrémité. La séquence suivie doit être pour accéder à la Nième dimension de la matrice pour toutes les valeurs des autres dimensions.

Cela doit être réalisé en accédant d'abord à la matrice avec toutes les dimensions mises à leurs valeurs d'indice initiales. Ensuite, la Nième dimension est incrémentée sur toutes ses valeurs d'indice. Lorsque la fin de la plage d'indices pour la Nième dimension est atteinte, la (N1)ième dimension est incrémentée et la Nième dimension est mis à sa valeur d'indice initiale. Ce processus est répété jusqu'à ce que toutes les dimensions de la matrice aient atteint la fin de leurs plages d'indices. Ainsi, la matrice est condensée dans le tampon de messages comme une matrice à une seule dimension. Le même mode opératoire est suivi pour décompresser la matrice unidimensionnelle en une matrice multidimensionnelle

La matrice bidimensionnelle

```
ARRAY1 [0..1 , 0..2] of UINT:= { { 1, 2, 3 },
          { 4, 5, 6 } }
```

conduit au train de données montré dans le Tableau 235 lorsqu'elle est condensée en une matrice à une seule dimension en suivant les règles de codage compact.

Tableau 235 – Exemple de codage compact d'un ARRAY à plusieurs dimensions

Numéro d'octet	1er	2e	3e	4e	5e	6e	7e	8e	9e	10e	11e	12e
ARRAY	01	00	02	00	03	00	04	00	05	00	06	00

5.1.3.9 Codage de structure

Le codage d'une structure doit utiliser les règles de codage pour les types de données pour chaque élément et concatène les éléments qui composent la structure.

Les valeurs codées des éléments de la structure doivent être codées dans le même ordre qu'elles sont déclarées dans la spécification de type ou de variable correspondante de l'ASN.1. Ces éléments peuvent être de n'importe quel type de données.

STRUCT::= SEQUENCE OF NetworkData – Peuvent être de types différents

Sachant que NetworkData peut être constitué des ElementaryData ou des DerivedData, une nouvelle spécification de type ou de variable peut être requise avant d'émettre les valeurs pour le STRUCT.

Supposons la définition de structure suivante:

newStruct::= SEQUENCE { BOOL,UINT,DINT}

L'insertion des valeurs {TRUE,'1234'H,'56789ABC'H} dans la structure donne le codage tel que spécifié dans le Tableau 236.

Tableau 236 – Exemple de codage compact d'une STRUCTURE

Numéro d'octet	1er	2e	3e	4e	5e	6e	7e
newStruct	01	34	12	BC	9A	78	56

5.1.3.10 Exemples de formats de données codées complexes

5.1.3.10.1 Généralités

Les exemples en 5.1.3.10.2 et 5.1.3.10.3 montrent comment des formats de données plus complexes doivent être condensés. L'exemple 5.1.3.10.2 montre la condensation d'une matrice de structures. L'exemple en 5.1.3.10.3 montre comment une structure ayant un élément matrice est condensée.

5.1.3.10.2 Exemple 1: le codage d'une matrice de structures:

```
STRUCT1::= SEQUENCE OF {
            UINT     ele1;
            USINT    ele2;
            USINT    ele3;
            USINT    ele4;
            UINT     ele5
}
  ARRAY1 [ 0..1 , 0..2 ] of STRUCT1:= {
            { { 1, 2, 3, 4, 5 }, { 6, 7, 8, 9, 10 },
              { 11, 12, 13, 14, 15 } },
            { { 15, 14, 13, 12, 11 }, { 10, 9, 8, 7, 6 },
              { 5, 4, 3, 2, 1 } } }
```

donne le train de données suivant:

```
[01][00][02][03][04][05][00] [06][00][07][08][09][0A][00]
[0B][00][0C][0D][0E][0F][00] [0F][00][0E][0D][0C][0B][00]
[0A][00][09][08][07][06][00] [05][00][04][03][02][01][00]
```

5.1.3.10.3 Exemple 2: le codage d'une structure ayant un élément matrice

```
STRUCT2::= SEQUENCE OF {
            UINT     ele1;
            ARRAY [ 0..2 ] of USINT array2;
            UINT     ele5;
}
STRUCT2:= { 1, { 2, 3, 4 }, 5 }
```

donne le train de données suivant:

```
[01][00] [02][03][04] [05][00]
```

5.2 Rapport relatif au type de données

5.2.1 Représentation de données d'objet

Des objets peuvent choisir de mettre en œuvre un mécanisme pour rendre compte du Data Type (type de données) d'un Attribut particulier ou pour émettre des informations relatives au type accompagnées des données effectives. Le Paragraphe 5.2 définit le moyen par lequel les informations de typage des données sont acheminées.

La spécification des informations relatives au type de données utilise la méthodologie de l'ASN.1 spécifiée dans l'ISO/CEI 8824-1 et dans l'ISO/CEI 8825-1 avec les optimisations définies par le réseau de commande pour coder la production de **DataTypeSpecification** définie en 4.2.4.

L'optimisation définie par le réseau de commande est que l'octet Length d'un type NULL n'est pas codé.

NOTE Par exemple, le codage de "abc [PRIVATE 1] IMPLICIT NULL" serait: 0xC1 (une étiquette sans octet de longueur).

5.2.2 Rapports relatifs aux types de données élémentaires

Les types de données élémentaires doivent être identifiés en utilisant les codes d'identification définis dans le Tableau 237. Ces codes doivent définir le codage des membres de primitive de la production de DataTypeSpecification. Le réseau de commande spécifie que les types NULL de l'ASN.1 ne doit pas rapporter l'octet de longueur de zéro (0).

Tableau 237 – Codes d'identification et descriptions des types de données élémentaires

Nom de type de données	Code de type de données (en hex)	Description de type de données
BOOL	C1	Booléen logique avec les valeurs TRUE et FALSE
SINT	C2	Valeur entière signée de 8 bits
INT	C3	Valeur entière signée de 16 bits
DINT	C4	Valeur entière signée de 32 bits
LINT	C5	Valeur entière signée de 64 bits
USINT	C6	Valeur entière non signée de 8 bits
UINT	C7	Valeur entière non signée de 16 bits
UDINT	C8	Valeur entière non signée de 32 bits
ULINT	C9	Valeur entière non signée de 64 bits
REAL	CA	Valeur en virgule flottante de 32 bits
LREAL	CB	Valeur en virgule flottante de 64 bits
STIME	CC	Information de temps synchrone

Nom de type de données	Code de type de données (en hex)	Description de type de données
DATE	CD	Information de date
TIME_OF_DAY	CE	Heure de la journée
DATE_AND_TIME	CF	Date et heure de la journée
STRING	D0	Chaîne de caractères (1 octet par caractère)
SWORD	D1	chaîne binaire 8 bits
WORD	D2	chaîne binaire 16 bits
DWORD	D3	chaîne binaire 32 bits
LWORD	D4	chaîne binaire 64 bits
STRING2	D5	chaîne de caractères (2 octets par caractère)
FTIME	D6	Durée (haute résolution)
LTIME	D7	Durée (longue)
ITIME	D8	Durée (courte)
STRINGN	D9	Chaîne de caractères (N octets par caractère)
SHORT_STRING	DA	chaîne de caractères (1 octet par caractère, indicateur de longueur 1 octet)
TIME	DB	Durée (millisecondes)
EPATH	DC	Segments de chemin
STRINGI	DE	Chaîne de caractères internationaux

5.2.3 Rapports relatifs aux types de données construits

5.2.3.1 Généralités

Le Paragraphe 5.2.3 expose en détail les moyens pour représenter les informations concernant la structure et la matrice dans la production DataTypeSpecification.

Le Tableau 238 spécifie les codes d'identification qui sont utilisés pour les divers types de données construites.

Tableau 238 – Codes d'identification et description des types de données construites

Nom de type de données	Code de type de données (en hex)	Description de type de données
Structure abrégée	A0	Structure identifiée par une valeur CRC de 16 bits
Matrice abrégée	A1	Matrice identifiée par une valeur CRC de 16 bits
Structure formelle	A2	Structure définie par la liste de types de membres
Matrice formelle	A3	Matrice définie par le type et la taille
	A4-A7	Réservé – ne pas utiliser
Structure de Formal Handle (identification formelle)	A8	Structure définie par le type de membre et par la liste d'identification des membres
	A9-BF	Réservé

5.2.3.2 Définition du type Structure

5.2.3.2.1 Généralités

Le réseau de commande définit trois méthodes différentes pour rapporter les définitions du type Structure:

- Codage formel (FormalStrucTypeSpec);
- Codage d'identification formelle (FormalHandleStrucTypeSpec);
- Codage abrégé (AbbreviatedStrucTypeSpec).

Le codage formel doit être utilisé pour fournir un rapport détaillé de la définition complète de la structure, y compris la définition complète de tous les types de données des composants (on peut obtenir le nombre de membres de la structure à partir de la position du composant). Le codage d'identification formelle est utilisé pour fournir un rapport détaillé de la définition complète de la structure, y compris la définition complète de tous les types de données des composants. Le codage abrégé doit être utilisé pour spécifier une forme plus courte de la définition de la structure. La forme plus courte ne doit pas inclure les types de données associés aux composants de la structure.

5.2.3.2.2 Codage formel pour les informations relatives au type de structure

Le Tableau 239 définit le format de codage formel pour les structures non imbriquées.

Tableau 239 – Définition du codage de structure formelle

Octet	Définition
0	Structure formelle (A2)
1	Longueur de la définition de structure en octets (n)
2 à (n+1)	Code de type de données (C0 à DF)

Les exemples en 5.2.3.2.3 et en 5.2.3.2.4 illustrent le codage formel pour les spécifications des types de structure. Il s'agit en fait d'un exemple du codage de la production de FormalStrucTypeSpec définie en 4.2.4.

5.2.3.2.3 Exemple 1

La Figure 21 illustre le codage de la définition de structure suivante.

```
STRUCT::= SEQUENCE OF { BOOL, UINT, DINT }
```

Type STRUCT	Longueur du type	Type de données (BOOL)	Type de données (UINT)	Type de données (DINT)
A2	03	C1	C7	C4
NOTE Les types NULL IMPLICIT issus de la production de DataTypeSpecification ne sont pas suivis d'un octet de longueur de zéro (0).				

Figure 21 – Exemple 1 de codage formel d'une spécification de type Structure

5.2.3.2.4 Exemple 2

La Figure 22 illustre le codage de la définition de structure suivante.

```
STRUCT_MAIN::= SEQUENCE OF { UINT, STRUCT_SUB, INT }

with subelement STRUCT_SUB defined as:

STRUCT_SUB::= SEQUENCE OF { UINT, SINT, INT }
```

Type Struct	Longueur du type	Composant						
		Type de données (UINT)	Type Struct	Longueur du type	Composant imbriqué			Type de données (INT)
					Type de données (UINT)	Type de données (SINT)	Type de données (INT)	
A2	07	C7	A2	03	C7	C2	C3	C3

Figure 22 – Exemple 2 de codage formel d'une spécification de type Structure

5.2.3.2.5 Codage formel pour Handle Structure Type Information (informations sur le type de structure d'identification)

Le Tableau 240 définit le format du codage formel pour les structures comportant des identifications.

Tableau 240 – Structure formelle avec définition du codage des identifications

Octet	Définition
0	Structure d'identification formelle (A8)
1	Longueur de la définition de structure en octets (n)
2	Longueur de l'identification (1, 2 ou 4 octets)
3	Code de type de données (C0 à DF) 1
4 à 4 + (longueur de l'identification – 1)	Identification (valeur à 1, 2 ou 4 octets attribuée par l'application) [a]

[a] Les membres supplémentaires sont représentés par des paires de codes de types de données et d'identifications.

Les deux exemples en 5.2.3.2.6 et en 5.2.3.2.7 illustrent le codage formel pour les spécifications des types de structure d'identification. Il s'agit en fait d'un exemple du codage de la production de FormalHandleStrucTypeSpec définie en 4.2.4.

5.2.3.2.6 Exemple 3

La Figure 23 illustre le codage de la définition de structure suivante.

```
STRUCT::= SEQUENCE OF { BOOL, UINT, DINT }
```

Type Structure	Longueur du type	Longueur de l'identification	Type de données (BOOL)	Identification de membre	Type de données (UINT)	Identification de membre	Type de données (DINT)	Identification de membre
A8	10	04	C1	D9 B9 74 3E	C7	F4 15 59 93	C4	0B 7B 24 A6

Figure 23 – Exemple 3 de codage formel d'une spécification de type Structure

5.2.3.2.7 Exemple 4

La Figure 24 illustre le codage de la définition de structure suivante.

```
STRUCT_MAIN::= SEQUENCE OF { UINT, STRUCT_SUB, INT }
```

le sous-élément STRUCT_SUB étant défini de la manière suivante:

```
STRUCT_SUB::= SEQUENCE OF { SINT, INT }
```

Type Structure	Longueur du type	Longueur de l'identification	Composant										
			Type de données (UINT)	Identification de membre	Type Struct	Longueur du type	Longueur de l'identification	Composant imbriqué				Type de données (INT)	Mancheron
								Type de données (SINT)	Identification de membre	Type de données (INT)	Identification de membre		
A8	0E	02	C7	90 1D	A8	05	01	C2	F8	C3	B2	C3	36 ED

Figure 24 – Exemple 4 de codage formel d'une spécification de type Structure

5.2.3.2.8 Codage abrégé pour les informations relatives au type de structure

Le Tableau 241 définit le format de codage abrégé pour les structures.

Tableau 241 – Définition du codage abrégé des structures

Octet	Définition
0	Structure abrégée (A0)
1	Longueur de la structure abrégée (02)
2 à 3	UINT contenant CRC

L'exemple en 5.2.3.2.9 illustre le codage abrégé pour les spécifications du type de structure. Il s'agit en fait d'un exemple du codage de la production de AbbreviatedStrucTypeSpec définie en 4.2.4.

L'UINT défini au sein de la production de AbbreviatedStrucTypeSpec doit être initialisé avec une valeur de contrôle de redondance cyclique (CRC) de 16 bits (voir CEI 61158-4-2). Ceci peut être utilisé par la logique d'application pour déterminer si, oui ou non, le format de la structure a changé. Le train d'octets utilisé pour produire le CRC est la spécification de type structure formellement codée (FormalStrucTypeSpec) réelle.

5.2.3.2.9 Exemple 5

La Figure 25 montre le codage abrégé de la définition de structure présentée en 5.2.3.2.8:

Struct Type	Type Length	UINT containing CRC	
A0	02	C7	26

Légende

Anglais	Français
Struct Type	Type Structure
Type Length	Longueur du type
UINT containing CRC	UINT contenant CRC

Figure 25 – Exemple 5 de codage abrégé d'une spécification de type Structure

NOTE Les algorithmes présentés dans la présente norme sont utilisés pour générer la valeur CRC de 0x26C7 issue de la spécification du type Formally Encoded (formellement codé): [A2][07][C7][A2][03] [C7][C2][C3][C3].

5.2.3.3 Définition du type Array

5.2.3.3.1 Généralités

Deux méthodes différentes pour rapporter les définitions du type Array (matrice) doivent être:

- Codage formel (FormalArrayTypeSpec);
- Codage abrégé (AbbreviatedArrayTypeSpec).

Le codage formel doit être utilisé pour fournir un rapport détaillé de la définition complète de la matrice, y compris le contenu de données et les dimensions de la matrice. Le codage abrégé doit être utilisé pour spécifier une forme plus courte de la définition de la matrice. Cette forme plus courte ne doit pas inclure les informations spécifiant les dimensions de la matrice.

5.2.3.3.2 Codage formel pour les informations relatives au type de matrice

Le Tableau 242 définit le format de codage abrégé pour les matrices.

Tableau 242 – Définition du codage formel de la matrice

Octet	Définition
0	Matrice formelle (A3)
1	Longueur de la définition de la matrice en octets (s)
2	Etiquette de borne inférieure (0x80)
3	Longueur de la borne inférieure (n – valeur 1, 2 ou 4)
4 à (4 + (n-1))	Borne inférieure
(4 + n)	Etiquette de borne supérieure (0x80)
(5 + n)	Longueur de la borne supérieure (n – valeur 1, 2 ou 4)
(6 + n) à ((6 + n) + (m-1))	Borne supérieure m
((6 + n) + m) à (6 + (s – 5))	Un des éléments suivants: 1 Code de type de données d'élément de matrice (C0 à DF) 2 Codage de matrice formel 3 Codage de structure formel 4 Codage de structure formel d'identification

Les exemples en 5.2.3.3.3 et en 5.2.3.3.4 illustrent le codage formel pour les spécifications des types de structure. Il s'agit en fait d'un exemple du codage de la production de FormalArrayTypeSpec définie en 4.2.4.

5.2.3.3.3 Exemple 1

La Figure 26 montre le codage formel de la définition de matrice suivante:

```
ARRAY1::= SEQUENCE OF { array_dimension_low_bound:= 0,
                        array_dimension_high_bound:= 9,
                        UINT
```

Matrice	Longueur du type	Etiquette de borne inférieure	Longueur de borne inférieure	Borne inférieure	Etiquette de borne supérieure	Longueur de borne supérieure	Borne supérieure	Type de données (UINT)
A3	07	80	01	00	81	01	09	C7

Figure 26 – Exemple 1 de codage formel d'une spécification de type Array

NOTE Les types NULL IMPLICIT issus de la production de DataTypeSpecification ne sont pas suivis d'un octet de longueur de zéro (0).

5.2.3.3.4 Exemple 2

La Figure 27 montre le codage de la définition de matrice suivante:

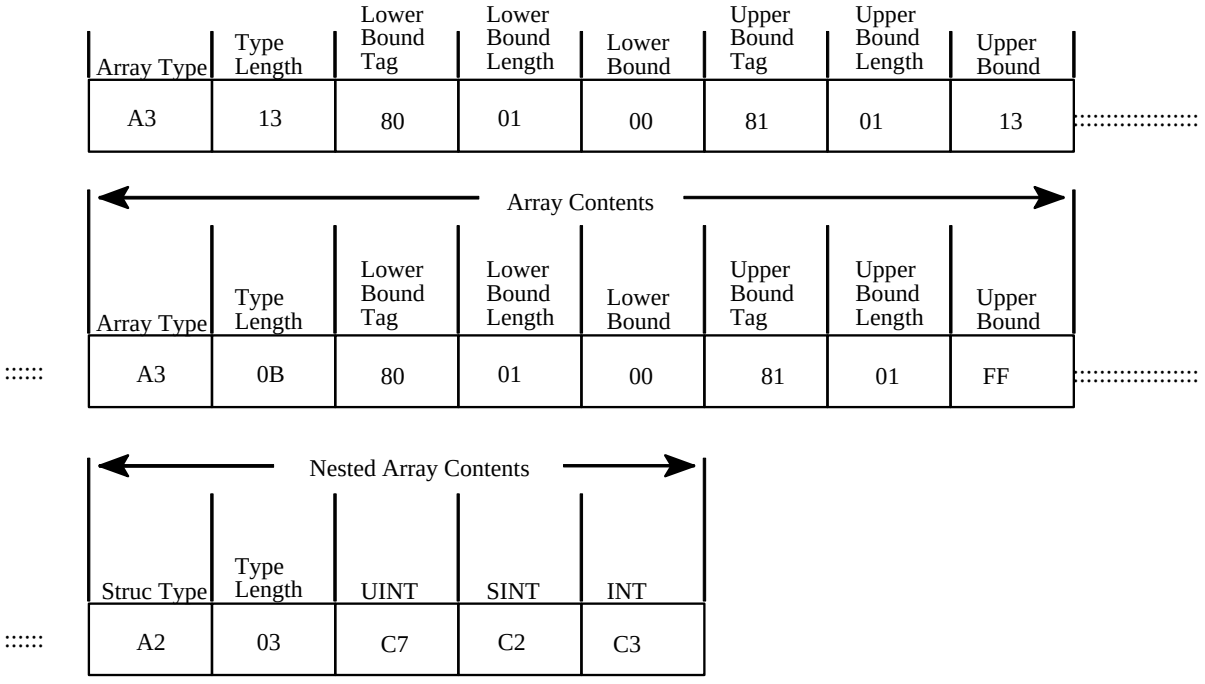
```
ARRAY1::= SEQUENCE OF { array_dimension_low_bound:= 0,
                        array_dimension_high_bound:= 19,
                      SEQUENCE OF { array_dimension_low_bound:= 0,
                          array_dimension_high_bound:= 255,
STRUCT_ELE } }
```

dans laquelle STRUCT_ELE est défini comme étant:

```
STRUCT_ELE::= SEQUENCE OF { UINT, SINT, INT }
```

Formal Encoding: [A3][13][80][01][00][81][01][13][A3][0B]80][01][00][81][01][FF][A2][03][C7][C2][C3]

Description:



Légende

Anglais	Français
Formal encoding	Codage formel

Anglais	Français
Array Type	Type Array
Type Length	Longueur de type
Lower Bound Tag	Etiquette de borne inférieure
Lower Bound Length	Longueur de borne inférieure
Lower Bound	Borne inférieure
Upper Bound Tag	Etiquette de borne supérieure
Upper Bound Length	Longueur de borne supérieure
Upper Bound	Borne supérieure
Array Contents	Contenu de la matrice
Nested Array Contents	Contenu imbriqué de la matrice
Struc Type	Type Structure

Figure 27 – Exemple 2 de codage formel d'une spécification de type Array

NOTE Les types NULL IMPLICIT issus de la production de DataTypeSpecification ne sont pas suivis d'un octet de longueur de zéro (0).

5.2.3.3.5 Codage d'étiquette abrégé pour les informations relatives au type array

Le Tableau 243 définit le format de codage abrégé pour les matrices.

Tableau 243 – Définition du codage abrégé des matrices

Octet	Définition
0	Matrice abrégée (A1)
1	Longueur de la matrice abrégée (n)
2 à (2+n-1)	Un des éléments suivants: 1 Code de type de données d'élément de matrice (C0 à DF) 2 Codage de matrice abrégé [a] 3 Codage de structure abrégé
[a] Chaque dimension de la matrice doit être représentée par une matrice abrégée.	

Le codage abrégé d'un type Array ne doit pas inclure les informations spécifiant les dimensions de la matrice. Il s'agit en fait d'un exemple du codage de la production de AbbreviatedArrayTypeSpec définie en 4.2.4.

5.2.3.3.6 Exemple 1

La Figure 28 montre le codage abrégé de la définition de matrice suivante:

```
ARRAY2::= SEQUENCE OF { array_dimension_low_bound:= 0,
                        array_dimension_high_bound:= 9,
                        UINT }
```

Type de groupement	Longueur du type	Type de données (UINT)
A1	01	C7

Figure 28 – Exemple 1 de codage abrégé d'une spécification de type Array

NOTE Les types NULL IMPLICIT issus de la production de DataTypeSpecification ne sont pas suivis d'un octet de longueur de zéro (0).

5.2.3.3.7 Exemple 2

La Figure 29 montre le codage abrégé de la définition de matrice suivante:

```
ARRAY::= SEQUENCE OF { array_dimension_low_bound:= 0,
                       array_dimension_high_bound:= 19,
                       SEQUENCE OF { array_dimension_low_bound:= 0,
                       array_dimension_high_bound:= 899,
STRUCT_ELE } }
```

dans laquelle STRUCT_ELE est défini comme étant:

```
STRUCT_ELE::= SEQUENCE OF { UINT, SINT, INT }
```

Type de groupement	Longueur du type	Type de groupement	Longueur du type	Structure (STRUCT_ELE)			
				Type Struct	Longueur du type	UINT contenant CRC	
A1	06	A1	04	A0	02	59	51

Figure 29 – Exemple 2 de codage abrégé d'une spécification de type Array

NOTE Les algorithmes présentés dans la présente norme sont utilisés pour générer la valeur CRC de 0x5159 issue de la spécification du type Formally Encoded (formellement codé): [A2][03][C7][C2][C3].

6 Structure des diagrammes d'états de protocole de la couche FAL

L'Article 6 décrit l'interface avec les services et les diagrammes protocolaires de couche FAL. Les conventions utilisées lors des descriptions figurent dans la CEI 61158-1.

Ce bus de terrain suit la structure indiquée pour le bus de terrain de Type 1 avec les caractéristiques spécifiques suivantes:

- Il n'y a pas de définition formelle de Diagramme de contexte AP (AP-Context Machine).
- Il y a un diagramme FSPM formellement défini qui sert d'interface entre l'utilisateur de la FAL et le diagramme ARPM.
- Il y a des diagrammes ARPM de deux types différents:
  1) un diagramme ARPM pour les relations d'application sans connexion;
  2) sept diagrammes ARPM pour les relations d'application orientées connexion.
- Les Machines DMPM sont définies à l'interface de la couche liaison de données (Data-link) prise en charge (Type 2 ou autres).

7 Diagramme d'états de contexte AP

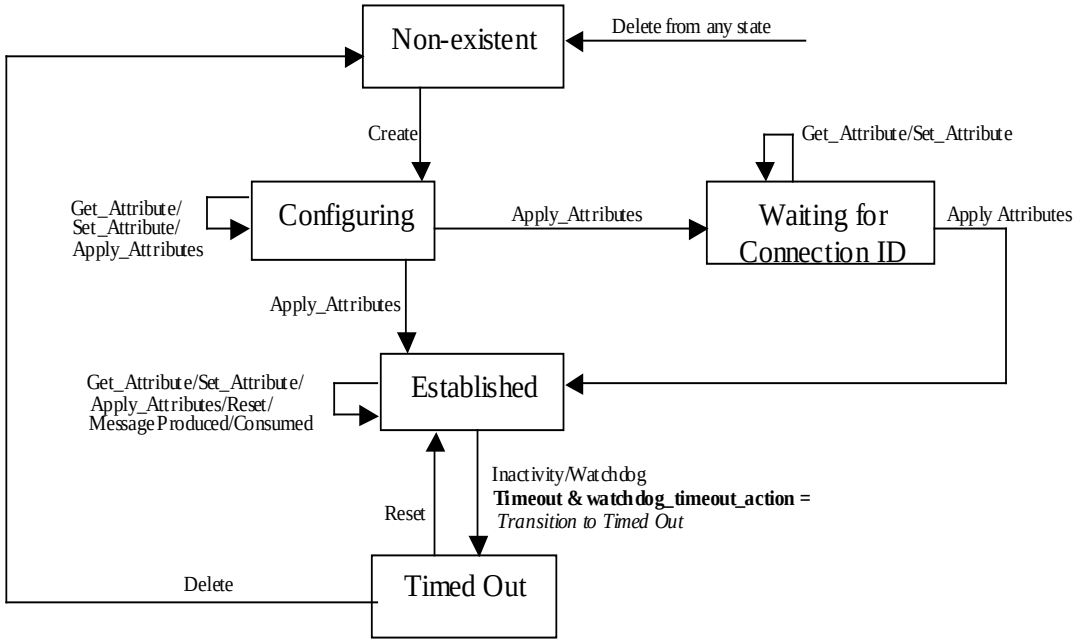
7.1 Vue d'ensemble

Le Type 2 prend en charge le diagramme d'états de contexte AP y spécifié, lorsque l'objet Connection est également pris en charge.

7.2 Diagramme d'état d'objet Connection

7.2.1 Comportement d'instance "I/O Connection"(Connexion E/S)

La Figure 30 donne une vue d'ensemble générale du comportement associé à un **objet I/O Connection** (attribut **instance_type** = E/S).



Légende

Anglais	Français
Delete from any state	Supprimer de n'importe quel état
Non-existent	Inexistant
Create	Créer
Delete	Supprimer
Configuring	En configuration
Waiting for Connection ID	En attente d'un identificateur de connexion
Established	Etabli
Message Produced/Consumed	Message produit/consommé
Timed Out	Temporisé
Inactivity/Watchdog Timeout & **watchdog_timeout_action** = Transition to Timed Out	Inactivity/Watchdog Temporisation et **watchdog_timeout_action** = Passage à Temporisé
Reset	Réinitialiser

Figure 30 – Diagramme de transitions d'états d'objet I/O Connection

Important: Le Tableau 244 fournit une State Event Matrix (Matrice d'événements d'états) pour un objet I/O Connection et il convient de baser les mises en œuvre sur ces informations. Cette State Event Matrix n'impose pas de règles concernant la logique interne spécifique à un produit. Une tentative d'accéder à la classe Connection ou à une instance d'objet Connection peut nécessiter de passer par une vérification spécifique à un produit. Il peut en résulter un scénario d'erreur qui n'est pas indiqué par la SEM dans le Tableau 244. Il peut aussi en résulter des indications supplémentaires spécifiques à un produit fournies par un objet Connection à une application et/ou à un objet d'application spécifique. Le point à garder à l'esprit est que l'objet Connection doit manifester le comportement visible de l'extérieur qui est spécifié par la SEM et les définitions d'attribut.

Tableau 244 – Matrice d'événements d'états d'objet I/O Connection

Evénement	Etat d'objet I/O Connection				
	Nonexistent (inexistant)	**Configuring (En configuration)**	**Waiting for Connection ID (En attente de l'ID de connexion)**	**Established (Etablie)**	**Timed Out (Temporisée)**
La classe Connection reçoit une demande Create.	La classe instancie un objet Connection. Mettre instance_type à I/O. Mettre tous les autres attributs aux valeurs par défaut. Passer à Configuring	Non applicable	Non applicable	Non applicable	Non applicable
La classe Connection reçoit une demande Delete.	Erreur: L'objet n'existe pas (Code d'erreur général 16hex)	Libérer toutes les ressources associées. Passer à Non-existent	Libérer toutes les ressources associées. Passer à Non-existent.	Libérer toutes les ressources associées. Passer à Non-existent.	Libérer toutes les ressources associées. Passer à Non-existent.
Set_Attribute _Single	Erreur: L'objet n'existe pas (Code d'erreur général 16hex)	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès ("Access Rules") Retourner la réponse	Si demande de modifier produced_connection_id ou consumed_connection_id, valider la valeur et gérer la demande. Retourner la réponse appropriée. S'il s'agit d'une demande d'accéder à un attribut autre que le produced_connection_id ou le consumed_connection_id, retourner une réponse d'erreur (Error Response) dont le Code d'erreur général est mis à 0Chex. (L'objet ne peut pas accomplir le service demandé dans son mode/état actuel.)	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès ("Access Rules") Retourner la réponse appropriée.	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès ("Access Rules") Retourner la réponse appropriée.
Get_Attribute _Single	Erreur: L'objet n'existe pas (Code d'erreur général 16hex)	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès ("Access Rules") Retourner la réponse	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès ("Access Rules") Retourner la réponse	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès ("Access Rules") Retourner la réponse	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès ("Access Rules") Retourner la réponse

Evénement	Etat d'objet I/O Connection				
	Nonexistent (inexistant)	**Configuring (En configuration)**	**Waiting for Connection ID (En attente de l'ID de connexion)**	**Established (Etablie)**	**Timed Out (Temporisée)**
Reset	Erreur: L'objet n'existe pas (Code d'erreur général 16hex)	Erreur: L'objet ne peut pas accomplir le service demandé dans son mode/état actuel. (Valeur du Code d'erreur général = 0Chex)	Erreur: L'objet ne peut pas accomplir le service demandé dans son mode/état actuel. (Valeur du Code d'erreur général = 0Chex)	Annuler le temporisateur Inactivity/Watchdog Timer courant. En utilisant la valeur de l'attribut expected_packet_rate, redémarrer le temporisateur Inactivity/Watchdog Timer. Une réponse de succès est retournée même si un Inactivity/Watchdog Timer n'est pas utilisé par l'objet Connection (Classe de transport Client 0, expected_packet_rate = 00Chex).	En utilisant la valeur de l'attribut expected_packet_rate, démarrer le temporisateur Inactivity/Watchdog Timer et revenir à l'état Established. Si l'attribut expected_packet_rate a été mis à zéro (0) alors que la connexion était dans l'état Timed Out (temporisée), revenir juste à l'état Established sans activer un temporisateur Inactivity/Watchdog Timer.
Apply_Attributes	Erreur: L'objet n'existe pas (Code d'erreur général 16hex)	Délivrer l'objet Connection à l'application qui valide les informations d'attribut. Si l'un ou l'autre des attributs connection_id (produit ou consommé) nécessite d'être configuré et ne peut pas être généré par ce module, exécuter toutes les autres étapes nécessaires pour configurer la connexion, retourner une réponse de réussite (Successful) et passer à l'état Waiting For Connection ID. L'incapacité de générer une valeur d'ID de connexion produite ou consommée N'EST PAS rapportée comme une erreur. Si tous les attributs sont configurés en toute validité, exécuter toutes les étapes nécessaires pour satisfaire à cette connexion, démarrer tous les temporisateurs requis et passer à l'état Established. Si une erreur est détectée, une réponse d'erreur est retournée et la connexion reste dans l'état Configuring[a] et, si une connexion client avec une valeur de déclencheur de production de 0 ou 1 ("Cyclic" ou "Change of State"), produire les données initiales.	Si l'un ou l'autre des attributs connection_id (produit ou consommé) nécessite encore d'être configuré et ne peut pas être généré par ce module, retourner une réponse de réussite (Successful) et rester dans l'état Waiting For Connection ID. L'incapacité de générer une valeur d'ID de connexion produite ou consommée N'EST PAS rapportée comme une erreur. Si l'attribut produced_connection_id et/ou l'attribut consumed_connection_id est/sont configuré(s) en toute validité, passer à l'état Established et retourner une réponse de succès[a] (Successful) et si une connexion client a une valeur de déclencheur de production 0 ou 1 (Cyclic ou Change of State), produire les données initiales.	Toutes les modifications ont lieu immédiatement dès que la connexion est passée à l'état Established. Retourner l'erreur: L'objet ne peut pas accomplir le service demandé dans son mode/état actuel. (Valeur du Code d'erreur général = 0Chex)	Erreur: L'objet ne peut pas accomplir le service demandé dans son mode/état actuel. (Valeur du Code d'erreur général = 0Chex)

Evénement	Etat d'objet I/O Connection				
	Nonexistent (inexistant)	**Configuring (En configuration)**	**Waiting for Connection ID (En attente de l'ID de connexion)**	**Established (Etablie)**	**Timed Out (Temporisée)**
Receive_Data	Non applicable	Rejeter le message	Rejeter le message	Si un message complet valide[b] a été reçu, réinitialiser le temporisateur Inactivity/Watchdog Timer[c] et délivrer le message I/O à l'application. Un objet Connection doit manifester le comportement visible de l'extérieur qui est associé à l'état actuel de ses attributs (voir Règles d'accès). S'il s'agit d'une partie fragmentée d'un message E/S, effectuer le traitement comme spécifié par le sous-réseau.	Rejeter le message
Send_Message	Non applicable	Retourner une erreur interne ne pas envoyer le message.	Retourner une erreur interne ne pas envoyer le message.	Emettre le message E/S complet ou le fragment comme requis par le sous-réseau. S'il s'agit d'une connexion client et l'attribut expected_packet_rate est différent de zéro, redémarrer le temporisateur Transmission Trigger Timer.	Retourner une erreur interne ne pas envoyer le message.
Inactivity/ Watchdog Timer expire	Non applicable	Non applicable	Non applicable	Examiner l'attribut watchdog_timeout_action de la connexion et accomplir l'action indiquée. Si l'attribut watchdog_timeout_action indique que la connexion doit rester dans l'état Established (*Auto Reset*), redémarrer immédiatement le temporisateur Inactivity/Watchdog Timer.	Non applicable

[a] Si la configuration indique qu'un ID de message a besoin d'être alloué et il n'existe pas d'ID de message disponible dans le Groupe de message spécifié, il est retourné une réponse d'erreur dont le Code d'erreur général indique *Resource Unavailable* (02hex). Si une valeur d'attribut d'objet Connection a passé avec succès la vérification de plage lorsqu'elle a été configurée initialement, mais la valeur d'attribut est en conflit avec un autre élément d'information dans le nœud lorsque la demande Apply est traitée, il est retourné une réponse d'erreur dont le Code d'erreur général est mis à *Valeur d'attribut non valide* (09hex) et dont le code supplémentaire (Additional Code) est mis à l'ID d'attribut de l'objet Connection *incriminé*.

[b] L'objet Connection vérifie que la longueur du message E/S reçu est inférieure ou égale à l'attribut consumed_connection_size avant de traiter le message. Si la longueur du message reçu est inférieure ou égale à l'attribut consumed_connection_size, l'objet I/O Connection réinitialise le temporisateur Inactivity/Watchdog, manifeste le comportement visible de l'extérieur indiqué par ses réglages d'attribut et délivre le message à l'Application. Si la longueur du message reçu est supérieure à l'attribut consumed_connection_size, l'objet I/O Connection doit immédiatement rejeter le message et interrompre tout traitement ultérieur. Il s'agit de la seule validation de contenu de message accomplie par l'objet I/O Connection. La validation ultérieure doit être accomplie par l'Application.

[c] Si un message fragmenté est en train d'être reçu, le temporisateur Inactivity/Watchdog Timer n'est pas réinitialisé tant que le message tout entier n'a pas été valablement reçu.

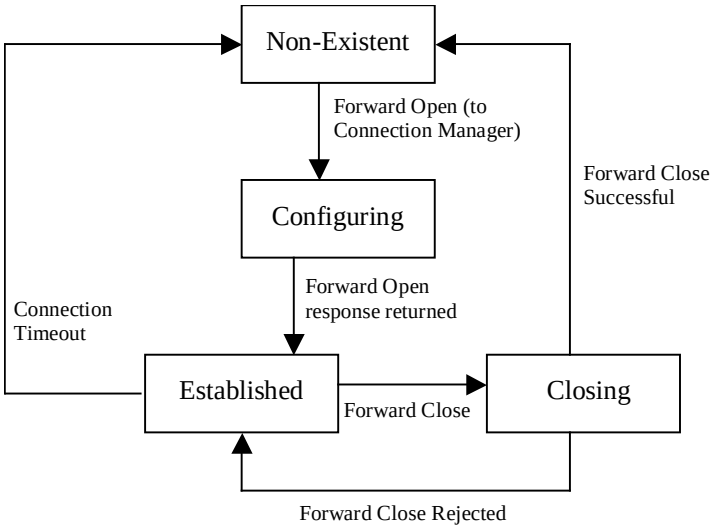
NOTE Les Règles d'accès d'attribut sont spécifiées dans la CEI 61158-5-2, Paragraphe 6.2.3.2.1.4.

Important: L'événement Receive_Data n'est délivré à une I/O Connection que lorsqu'il est reçu un message dont le champ Connection Identifier concorde avec l'attribut consumed_connection_id. S'il est reçu un message dont le champ Connection Identifier ne concorde avec aucun attribut consumed_connection_id d'un objet Connection Established, le message est rejeté. Les identificateurs de connexion (Connection Identifier) sont spécifiques à un type de sous-réseau.

Si une mise en œuvre détecte qu'elle ne prend pas en charge un Explicit Messaging Service indiqué au Tableau 244, une réponse d'erreur (Error Response) qui spécifie un code d'erreur générale 08 (General Error Code 08) est retourné.

7.2.2 Comportement d'instance de "Bridged Connection"

La Figure 31 fournit une vue d'ensemble générale du comportement associé à un objet **Bridged** Connection (attribut **instance_type** = Bridged). Les connexions pontées servent à effectuer des connexions offlink (hors liaison). La messagerie tant E/S qu'explicite peut être accomplie avec l'aide de ce type de connexion. La définition de l'objet Connection Manager donne plus de détails sur ces types de connexion.



Légende

Anglais	Français
Non-Existent	Inexistant
Forward Open (to Connection Manager)	Transmettre Ouvrir (au gestionnaire de connexion)
Forward Close Successful	Transmettre Fermer a abouti
Configuring	En configuration
Forward Open response returned	Réponse à Transmettre Ouvrir retournée
Connection Timeout	Temporisation de connexion
Established	Etablie
Forward Close	Transmettre Fermer
Closing	En cours de fermeture
Forward Close Rejected	Transmettre Fermer rejeté

Figure 31 – Diagramme de transitions d'états d'un objet Connection "Bridged"

Le Tableau 245 fournit une State Event Matrix détaillée pour un objet Connection "Bridged".

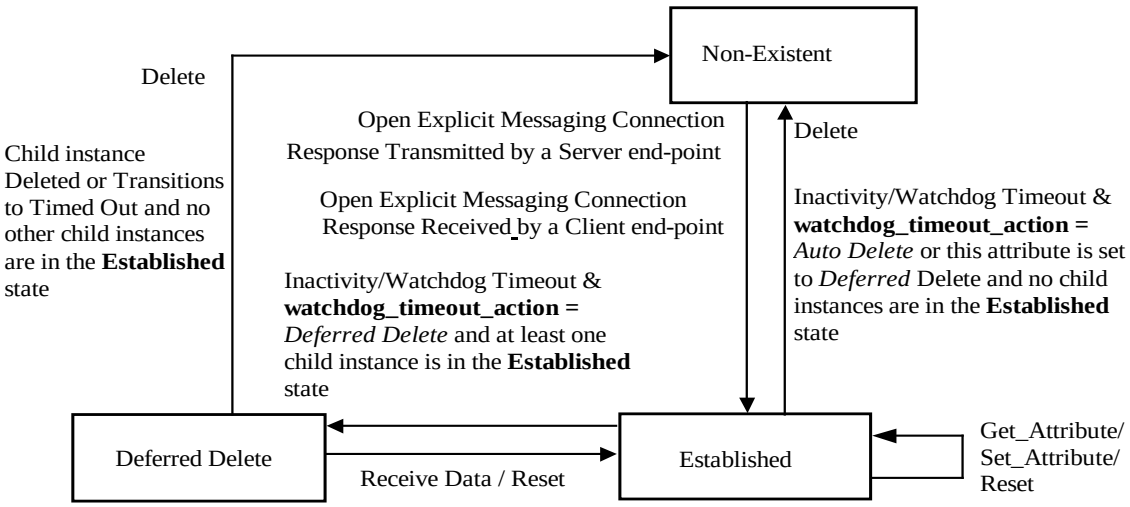
Tableau 245 – Matrice d'événements d'états d'une connexion "Bridged"

Evénement	Etat de l'objet Connection "Bridged"			
	Non-existent (Inexistant)	**Configuring (En configuration)**	**Established (Etablie)**	**Closing (En cours de fermeture)**
Le Connection Manager reçoit une demande Forward_Open.	La classe Connection instancie un objet Connection. Mettre **instance_type** à **Bridged**. Mettre les attributs à la valeur délivrée par le service Forward_Open ou à la valeur par défaut pour les connexions pontées. Passer à **Configuring**.	Non applicable	Non applicable	Non applicable
Le Connection Manager reçoit la notification que l'établissement de connexion à la cible est achevé.	Non applicable	Passer à **Established**	Non applicable	Non applicable
Le Connection Manager reçoit une demande Forward_Close.	Non applicable	Ignorer l'événement	Passer à **Closing**	Ignorer l'événement
Le Connection Manager reçoit une réponse Forward_Close.	Non applicable	Libérer toutes les ressources et passer à **Non-existent**.	Libérer toutes les ressources et passer à **Non-existent**.	Libérer toutes les ressources et passer à **Non-existent**.
Delete	Erreur: L'objet n'existe pas (Code d'erreur général 16_{hex})	Libérer toutes les ressources et passer à **Non-existent**.	Libérer toutes les ressources et passer à **Non-existent**.	Libérer toutes les ressources et passer à **Non-existent**.
Get_Attribute_Single	Erreur: L'objet n'existe pas (Code d'erreur général 16_{hex})	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès ("Access Rules") Retourner la réponse appropriée.	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès ("Access Rules") Retourner la réponse appropriée.	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès ("Access Rules") Retourner la réponse appropriée.
Set_Attribute_Single	Erreur: L'objet n'existe pas (Code d'erreur général 16_{hex})			
Reset	Erreur: L'objet n'existe pas (Code d'erreur général 16_{hex})	Erreur: Service non pris en charge (Code d'erreur général 08_{hex})	Erreur: Service non pris en charge (Code d'erreur général 08_{hex})	Erreur: Service non pris en charge (Code d'erreur général 08_{hex})
Apply_Attributes	Erreur: L'objet n'existe pas (Code d'erreur général 16_{hex})	Erreur: Service non pris en charge (Code d'erreur général 08_{hex})	Erreur: Service non pris en charge (Code d'erreur général 08_{hex})	Erreur: Service non pris en charge (Code d'erreur général 08_{hex})
Receive_Data	Non applicable	Ignorer l'événement	Invoquer le service envoi de l'objet Connection sur le port de destination, en transmettant les données reçues.	Ignorer l'événement

Evénement	Etat de l'objet Connection "Bridged"			
	Non-existent (Inexistant)	**Configuring (En configuration)**	**Established (Etablie)**	**Closing (En cours de fermeture)**
Send_Message	Non applicable	Ignorer l'événement	Envoyer les données sur le sous-réseau.	Ignorer l'événement
Le temporisateur Inactivity/Watchdog Timer expire	Non applicable	Non applicable	Libérer toutes les ressources et passer à **Non-existent**.	Libérer toutes les ressources et passer à **Non-existent**.
NOTE Les Règles d'accès d'attribut sont spécifiées dans la CEI 61158-5-2, Paragraphe 6.2.3.2.1.4.				

7.2.3 Comportement de l'instance de connexion de messagerie explicite

La Figure 32 donne une vue d'ensemble générale du comportement associé à un objet **Connection "Explicit Messaging"** (attribut **instance_type** = Explicit Messaging).



Légende

Anglais	Français
Non-Existent	Inexistant
Established	Etablie
Inactivity/Watchdog Timeout & watchdog_timeout_action = Auto Delete or this attribute is set to Deferred Delete and no child instances are in the Established state	Temporisation Inactivity/Watchdog et watchdog_timeout_action = suppression automatique ou cet attribut est mis à Suppression différée et aucune instance enfant ne se trouve à l'état Etablie
Reset	Réinitialiser
Open Explicit Messaging Connection Response Transmitted by a Server end-point	Un point d'extrémité serveur a émis une réponse à une demande d'ouvrir une connexion de messagerie explicite
Open Explicit Messaging Connection Response Received by a Client end-point	Un point d'extrémité client a reçu une réponse à une demande d'ouvrir une connexion de messagerie explicite
Delete	Supprimer
Deferred Delete	Suppression différée
Inactivity/Watchdog Timeout & watchdog_timeout_action = Deferred Delete and at least one child instance is in the Established state	Temporisation Inactivity/Watchdog et watchdog_timeout_action = Suppression différée et au moins une instance enfant se trouve à l'état Etablie
Child instance Deleted or Transitions to Timed Out and no other child instances are in the	L'instance enfant est supprimée ou passe à Temporisée et aucune autre instance enfant ne

Anglais	Français
Established state	se trouve à l'état Etablie
Receive Data / Reset	Recevoir des données/Réinitialiser

Figure 32 – Diagramme de transitions d'états d'un objet Connection "Explicit Messaging"

Le Tableau 246 fournit une State Event Matrix détaillée pour un objet Connection "Explicit Messaging". Il convient de baser les mises en œuvre sur les informations du Tableau 246.

Tableau 246 – Matrice d'événements d'états d'une connexion "Explicit Messaging"

Evénement	Etat de l'objet Connection "Explicit Messaging"		
	Nonexistent (inexistant)	**Established (Etablie)**	**Deferred Delete (Suppression différée)**
L'UCMM reçoit une demande/réponse d'ouvrir une connexion de messagerie explicite (Open Explicit Messaging Connection) et invoque le service Create de la classe de connexion	Si possible, la classe instancie l'objet Connection. Mettre l'attribut **instance_type** à *Explicit Messaging*[a]. Les autres attributs sont automatiquement configurés à l'aide des informations contenues dans la demande/réponse de "Open Explicit Messaging Connection". Passer à **Established.** Si demande reçue, émettre la réponse "Open Explicit Messaging Connection".	Non applicable	Non applicable
L'UCMM reçoit une demande Close et invoque le service Delete de la classe de connexion.	Erreur: L'objet n'existe pas (Code d'erreur général 16hex)	Libérer toutes les ressources associées. Passer à **Non-existent**.	Libérer toutes les ressources associées. Passer à **Non-existent**.
La classe Connection reçoit une demande Delete.	Erreur: L'objet n'existe pas (Code d'erreur général 16hex)	Libérer toutes les ressources associées. Passer à **Non-existent**.	Libérer toutes les ressources associées. Passer à **Non-existent**.
Set_Attribute_Single	Erreur: L'objet n'existe pas (Code d'erreur général 16hex)	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès ("Access Rules") Retourner la réponse appropriée.	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès ("Access Rules") Retourner la réponse appropriée.
Get_Attribute_Single	Erreur: L'objet n'existe pas (Code d'erreur général 16hex)	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès ("Access Rules") Retourner la réponse appropriée.	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès ("Access Rules") Retourner la réponse appropriée.
Reset	Erreur: L'objet n'existe pas (Code d'erreur général 16hex)	Annuler le temporisateur Inactivity/Watchdog Timer courant. En utilisant la valeur de l'attribut **expected_packet_rate**, redémarrer le temporisateur Inactivity/Watchdog Timer.	En utilisant la valeur de l'attribut **expected_packet_rate**, redémarrer le temporisateur Inactivity/Watchdog Timer et revenir à l'état Established.

Evénement	Etat de l'objet Connection "Explicit Messaging"		
	Nonexistent (inexistant)	**Established (Etablie)**	**Deferred Delete (Suppression différée)**
Apply_Attributes	Erreur: L'objet n'existe pas (Code d'erreur général 16hex)	Erreur: L'objet ne peut pas accomplir le service demandé dans son mode/état actuel. (Valeur du Code d'erreur général = 0Chex)	Erreur: L'objet ne peut pas accomplir le service demandé dans son mode/état actuel. (Valeur du Code d'erreur général = 0Chex)
Receive_Data	Non applicable	Si un message ou fragment de message valide a été reçu, réinitialiser le temporisateur Inactivity/Watchdog Timer[b]. Traiter/stocker le fragment ou bien gérer le message explicite.	Si un message ou fragment de message valide a été reçu, redémarrer le temporisateur Inactivity/Watchdog Timer[b] et revenir à l'état **Established**. Traiter/stocker le fragment ou bien gérer le message explicite.
Send_Message	Non applicable	Examiner la longueur du message à émettre et, le cas échéant, accomplir une série fragmentée d'émissions. Autrement, émettre le message explicite complet.	Examiner la longueur du message à émettre et, le cas échéant, accomplir une série fragmentée d'émissions. Autrement, émettre le message explicite complet.
Le temporisateur Inactivity/Watchdog Timer expire	Non applicable	Si l'attribut **watchdog_timeout_action** est mis à *Auto Delete* ou est mis à *Deferred Delete* et aucune instance de connexion enfant n'est dans l'état **Established**, libérer toutes les ressources associées et passer à **Non-existent**. Si l'attribut **watchdog_timeout_action** est mis à *Deferred Delete* et au moins une instance de connexion enfant est dans l'état **Established**, passer à **Deferred** Delete	Non applicable
Instance de connexion enfant Deleted (supprimée) ou passage à Timed Out	Non applicable	Ignorer l'événement	Si aucune autre instance de connexion enfant n'est dans l'état **Established**, libérer toutes les ressources associées et passer à **Nonexistent**.

[a] Si la configuration indique qu'une ressource de connexion a besoin d'être allouée et il n'existe pas de ressource disponible, il est retourné une réponse d'erreur dont le Code d'erreur général indique "Resource Unavailable" (Ressource non disponible) (02hex).

[b] Sur CP 2/3, le champ MAC ID dans l'en-tête de message de tous les messages explicites et/ou fragments de message est examiné. Si l'ID de MAC de la destination est spécifiée dans l'ID de connexion (CAN Identifier Field), l'ID de MAC de source de l'autre point d'extrémité doit être spécifié dans l'en-tête de message. Si l'ID de MAC de la source est spécifiée dans l'ID de connexion, l'ID de MAC du module de réception doit être spécifié dans l'en-tête de message. Si l'une de ces vérifications échoue, le temporisateur Inactivity/Watchdog Timer n'est pas réinitialisé et le message/fragment de message est rejeté.

NOTE Les Règles d'accès d'attribut sont spécifiées dans la CEI 61158-5-2, Paragraphe 6.2.3.2.1.4.

Important: L'événement Receive_Data n'est délivré à une connexion "Explicit Messaging " que lorsqu'il est reçu un message dont l'ID de connexion concorde avec l'attribut consumed_connection_id. S'il est reçu un message dont l'id de connexion ne concorde avec aucun attribut consumed_connection_id d'un objet Connection "Established", le message est rejeté.

Si une mise en œuvre détecte qu'elle ne prend pas en charge un Explicit Messaging Service indiqué au Tableau 246, une réponse d'erreur (Error Response) qui spécifie un code d'erreur générale 08 (General Status Code 08) est retourné.

8 Diagramme protocolaire de services de la FAL (FSPM)

8.1 Généralités

Ce diagramme d'états protocolaires de services FAL de bus de terrain met l'utilisateur de la FAL en correspondance avec les services d'objets de communication internes à la FAL.

8.2 Définitions de primitive

Les objets au sein du diagramme FSPM doivent fournir les services suivants:

Get_Attribute_All	Lit les valeurs de tous les attributs de la classe ou instance d'objet spécifiée.
Set_Attribute_All	Ecrit dans tous les attributs de la classe ou instance d'objet spécifiée les valeurs spécifiées.
Get_Attribute_List	Lit les valeurs de la liste spécifiée d'attributs de la classe ou de l'instance d'objet spécifiée.
Set_Attribute_List	Ecrit dans la liste spécifiée d'attributs de la classe ou de l'instance d'objet spécifiée les valeurs spécifiées.
Get_Attribute_Single	Lit la valeur de l'attribut spécifié de la classe ou de l'instance d'objet spécifiée.
Set_Attribute_Single	Ecrit dans l'attribut spécifié de la classe ou de l'instance d'objet spécifiée la valeur spécifiée.
Reset	Réinitialise la classe ou l'instance d'objet spécifiée.
Create	Crée une instance de la classe d'objets spécifiée.
Delete	Supprime une instance de la classe d'objets spécifiée.
Start	Démarre l'exécution de l'objet spécifié.
Stop	Arrête l'exécution de l'objet spécifié.
Find_Next_Object_Instance	Trouve l'identificateur de la prochaine instance non utilisée de la classe d'objets spécifiée.
NOP	Déclenche la réponse NOP correspondante issue de l'objet spécifié.
Apply_Attributes	Amène les valeurs d'attribut en cours à devenir actives dans l'objet spécifié.
Save	Sauvegarde le contenu des attributs de l'objet spécifié.
Restore	Restaure le contenu des attributs de l'objet spécifié.
Get_Member	Renvoie les informations sur le/les membre(s) depuis un attribut.
Set_Member	Ecrit dans un attribut les informations concernant le ou les membres
Insert_Member	Insère un/des membre(s) dans un attribut.
Remove_Member	Supprime un/des membre(s) d'un attribut.
Group_Sync	Vérifie que chaque membre d'un groupe est synchronisé au temps du système.

Pour les définitions détaillées de ces services, se référer à la CEI 61158-5-2.

Les primitives des services communs échangés entre l'UCMM et l'utilisateur de la FAL sont montrées dans le Tableau 248 et dans le Tableau 249.

Tous les services ont les paramètres communs ci-après, montrés dans le Tableau 247.

Tableau 247 – Primitives émises par l'utilisateur de la FAL vers le diagramme FSPM

Paramètres communs	Req	Ind	Rsp	Cnf
Argument	M			
AREP	M			
Local	S			
UCMM Record identifier	S			
Transport identifier	S			
Receiver/Server Local ID		M		
Path	M	C		
Routing Path	M			
Add. Path	U	U(=)		
Port ID		M		
Result (+)			S	S(=)
AREP				M=
Receiver/Server Local ID			M=	
Service status			M	M(=)
Result (-)			S	S(=)
AREP				M=
Receiver/Server Local ID			M=	
Service status			M	M(=)

Seuls les paramètres arguments complémentaires des paramètres communs sont montrés dans le Tableau 248 et dans le Tableau 249.

Les paramètres complémentaires des primitives de services "indication" sont les même ceux de la primitive "request" correspondante.

Les paramètres complémentaires des primitives de services "confirmation" sont les même ceux de la primitive "response" correspondante.

Tableau 248 – Primitives émises par l'utilisateur de la FAL vers le diagramme FSPM

Nom de la primitive	Source	Paramètres complémentaires	Fonctions
Get_attribute_all.req	Utilisateur de la FAL	Aucun	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour fournir des valeurs de tous les attributs de la classe ou de l'instance d'objet spécifiée.
Set_attribute_all.req	Utilisateur de la FAL	Attribute Data	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour écrire des valeurs spécifiées de tous les attributs de la classe ou de l'instance d'objet spécifiée.
Get_attribute_list.req	Utilisateur de la FAL	Attribute Count Attribute List	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour fournir des valeurs d'une liste spécifiée d'attributs de la classe ou de l'instance d'objet spécifiée.
Set_attribute_list.req	Utilisateur de la FAL	Attribute Count Attribute Data	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour écrire des valeurs spécifiées d'une liste spécifiée d'attributs de la classe ou de l'instance d'objet spécifiée.

Nom de la primitive	Source	Paramètres complémentaires	Fonctions
Get_attribute_single.req	Utilisateur de la FAL	Aucun	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour fournir des valeurs de l'attribut spécifié de la classe ou de l'instance d'objet spécifiée.
Set_attribute_single.req	Utilisateur de la FAL	Attribute Data	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour écrire dans l'attribut spécifié de la classe ou de l'instance d'objet spécifiée les valeurs spécifiées.
Reset.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL pour réinitialiser la classe ou l'instance spécifiée de l'objet cible
Create.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour créer une instance de la classe d'objets spécifiée.
Delete.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour supprimer une instance de la classe d'objets spécifiée.
Start.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour démarrer une instance de la classe d'objets spécifiée.
Stop.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour arrêter une instance de la classe d'objets spécifiée.
Find_next_object_ instance.req	Utilisateur de la FAL	Maximum Returned Values	Achemine une demande de l'utilisateur de la FAL vers un appareil cible pour trouver l'identificateur de la prochaine instance non utilisée de la classe d'objets spécifiée.
NOP.req	Utilisateur de la FAL	Aucun	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour renvoyer une réponse NOP correspondante.
Apply_Attributes.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour activer des valeurs d'attribut en cours.
Save.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour sauvegarder ses contenus d'attribut.
Restore.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour restaurer ses contenus d'attribut.
Get_Member.req	Utilisateur de la FAL	Aucun	Achemine une demande émise par l'utilisateur de la FAL à un objet cible pour fournir des informations concernant le ou les membres qui se trouvent dans l'attribut spécifié
Set_Member.req	Utilisateur de la FAL	Member data	Achemine une demande émise par l'utilisateur de la FAL à un objet cible pour écrire des informations concernant le ou les membres dans l'attribut spécifié
Insert_Member.req	Utilisateur de la FAL	Member Data (facultatif)	Achemine une demande émise par l'utilisateur de la FAL à un objet cible pour insérer un ou plusieurs membres dans l'attribut spécifié
Remove_Member.req	Utilisateur de la FAL	Aucun	Achemine une demande émise par l'utilisateur de la FAL à un objet cible pour supprimer de l'attribut spécifié un ou plusieurs membres
Group_Sync.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour vérifier que chaque membre d'un groupe est synchronisé au temps du système.
Get_attribute_all.rsp	Utilisateur de la FAL	(+) Attribute Data (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet cible pour fournir des valeurs de tous les attributs de la classe ou de l'instance d'objet spécifiée.

Nom de la primitive	Source	Paramètres complémentaires	Fonctions
Set_attribute_all.rsp	Utilisateur de la FAL	(+) None (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet cible pour écrire des valeurs spécifiées de tous les attributs de la classe ou de l'instance d'objet spécifiée.
Get_attribute_list.rsp	Utilisateur de la FAL	(+) Attribute Count Attribute Data (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet cible pour fournir des valeurs d'une liste spécifiée d'attributs de la classe ou de l'instance d'objet spécifiée.
Set_attribute_list.rsp	Utilisateur de la FAL	(+) Attribute_count Attribute Status List (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet cible pour écrire des valeurs spécifiées d'une liste spécifiée d'attributs de la classe ou de l'instance d'objet spécifiée.
Get_attribute_single.rsp	Utilisateur de la FAL	(+) Attribute Data (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet cible pour fournir des valeurs de l'attribut spécifié de la classe ou de l'instance d'objet spécifiée.
Set_attribute_single.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet cible pour écrire dans l'attribut spécifié de la classe ou de l'instance d'objet spécifiée les valeurs spécifiées.
Reset.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet cible pour confirmer que l'instance de la classe d'objets spécifiée a été réinitialisée.
Create.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet cible pour confirmer que l'instance de la classe d'objets spécifiée a été créée.
Delete.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet cible pour confirmer que l'instance de la classe d'objets spécifiée a été supprimée.
Start.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet cible pour confirmer que l'instance de la classe d'objets spécifiée a été démarrée.
Stop.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet cible pour confirmer que l'instance de la classe d'objets spécifiée a été arrêtée.
Find_next_object_nce.rsp	Utilisateur de la FAL	(+) Number Of List Members Instance ID List (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un appareil cible pour trouver l'identificateur de la prochaine instance non utilisée de la classe d'objets spécifiée.
NOP.rsp	Utilisateur de la FAL	Aucun	Retourne une réponse de l'utilisateur de la FAL vers un objet cible pour confirmer que le NOP a été reçu.
Apply_Attributes.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet cible pour confirmer que les valeurs d'attribut en cours ont été activées.
Save.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet cible pour confirmer que ses contenus d'attribut ont été sauvegardés.
Restore.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet cible pour confirmer que ses contenus d'attribut ont été restaurés.
Get_Member.rsp	Utilisateur de la FAL	(+) Member ID Member Data (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL à un objet cible pour fournir à un ou à plusieurs membres des informations qui se trouvent dans l'attribut spécifié
Set_Member.rsp	Utilisateur de la FAL	(+) Member ID (conditionnel) Member Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL à un objet cible pour écrire dans l'attribut spécifié des informations concernant le ou les membres

Nom de la primitive	Source	Paramètres complémentaires	Fonctions
Insert_Member.rsp	Utilisateur de la FAL	(+) Member ID Member Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL à un objet cible pour insérer un ou plusieurs membres dans l'attribut spécifié
Remove_Member.rsp	Utilisateur de la FAL	(+) Member ID Member Data (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL à un objet cible pour supprimer de l'attribut spécifié un ou plusieurs membres
Group_Sync.rsp	Utilisateur de la FAL	(+) IsSynchronized Object Specific Data (facultatif) (-) Specific Status Code	Achemine une demande de l'utilisateur de la FAL vers un objet cible pour vérifier que chaque membre d'un groupe est synchronisé au temps du système.

Tableau 249 – Primitives émises par le diagramme FSPM vers l'utilisateur de la FAL

Nom de la primitive	Source	Paramètres complémentaires	Fonctions
Get_attribute_all.ind	FSPM	idem que dans la primitive .req	Indique la réception de Get_attribute_all.req
Set_attribute_all.ind	FSPM	idem que dans la primitive .req	Indique la réception de Set_attribute_all.req
Get_attribute_list.ind	FSPM	idem que dans la primitive .req	Indique la réception de Get_attribute_list.req
Set_attribute_list.ind	FSPM	idem que dans la primitive .req	Indique la réception de Set_attribute_list.req
Get_attribute_single.ind	FSPM	idem que dans la primitive .req	Indique la réception de Get_attribute_single.req
Set_attribute_single.ind	FSPM	idem que dans la primitive .req	Indique la réception de Set_attribute_single.req
Reset.ind	FSPM	idem que dans la primitive .req	Indique la réception de Reset.req
Create.ind	FSPM	idem que dans la primitive .req	Indique la réception de Create.req
Delete.ind	FSPM	idem que dans la primitive .req	Indique la réception de Delete.req
Start.ind	FSPM	idem que dans la primitive .req	Indique la réception de Start.req
Stop.ind	FSPM	idem que dans la primitive .req	Indique la réception de Stop.req
Find_next_object_instance.ind	FSPM	idem que dans la primitive .req	Indique la réception de Find_next_object_instance.req
NOP.ind	FSPM	idem que dans la primitive .req	Indique la réception de NOP.req
Apply_Attributes.ind	FSPM	idem que dans la primitive .req	Indique la réception de Apply_Attributes.req
Save.ind	FSPM	idem que dans la primitive .req	Indique la réception de Save.req
Restore.ind	FSPM	idem que dans la primitive .req	Indique la réception de Restore.req
Get_Member.ind	FSPM	idem que dans la primitive .req	Indique la réception de Get_Member.req
Set_Member.ind	FSPM	idem que dans la primitive .req	Indique la réception de Set_Member.req
Insert_Member.ind	FSPM	idem que dans la primitive .req	Indique la réception de Insert_Member.req
Remove_Member.ind	FSPM	idem que dans la primitive .req	Indique la réception de Remove_Member.req

Nom de la primitive	Source	Paramètres complémentaires	Fonctions
Group_Sync.ind	FSPM	idem que dans la primitive .req	Indique la réception de Group_Sync.req
Get_attribute_all.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Get_attribute_all.rsp
Set_attribute_all.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Set_attribute_all.rsp
Get_attribute_list.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Get_attribute_list.rsp
Set_attribute_list.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Set_attribute_list.rsp
Get_attribute_signle.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Get_attribute_single.rsp
Set_attribute_single.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Set_attribute_single.rsp
Reset.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Reset.rsp
Create.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Create.rsp
Delete.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Delete.rsp
Start.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Start.rsp
Stop.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Stop.rsp
Find_next_object_instance.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Find_next_object_instance.rsp
NOP.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de NOP.rsp
Apply_Attributes.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Apply_Attributes.rsp
Save.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Save.rsp
Restore.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Restore.rsp
Get_Member.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Get_Member.rsp
Set_Member.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Set_Member.rsp
Insert_Member.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Insert_Member.rsp
Remove_Member.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Remove_Member.rsp
Group_Sync.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Group_Sync.rsp

8.3 Paramètres des primitives

Les paramètres utilisés avec les primitives échangées entre la machine FSPM et la machine ARPM sont décrits dans le Tableau 250.

Tableau 250 – Paramètres utilisés avec les primitives échangées entre l'utilisateur de la FAL et le diagramme FSPM

Nom de paramètre	Description
AREP: Local identifier UCMM Record identifier Transport identifier	Identifie l'entité à utiliser pour acheminer le service. Ce paramètre peut utiliser un identificateur spécialisé associé à une entité locale, l'identificateur d'un enregistrement de transaction UCMM créé au préalable, ou l'identificateur de transport retourné par le processus d'établissement de connexion et associé à l'AR sélectionnée.
Receiver/Server Local ID	Généré par l'ASE Message Router du nœud répondeur. Identifie localement cette invocation du service. Il est utilisé pour associer des réponses de services à des indications.
Path: Routing Path Additional path	Dans la demande, il spécifie l'APO ("Application Process Object", objet de processus d'application) de la FAL ou l'élément d'APO de FAL qui est la destination de la demande de service. Dans l'indication, il contient seulement les segments au-delà du segment de classe logique issu de la demande de service, c'est-à-dire les informations complémentaires facultatives pour l'APO cible (Add.Path).
Port ID	Indique sur quel port de l'appareil l'indication de service est arrivée.
Service status: Status Code (obligatoire) Extended Status (conditionnel)	Donne des informations relatives au résultat d'exécution du service. Il est retourné dans toutes les primitives "confirmed service response" (+ et -). Status code (code de statut) indique le type d'erreur (voir la CEI 61158-5-2 pour des détails). Extended status code (code de statut étendu) donne des détails du statut (voir la CEI 61158-5-2 pour des détails).
Attribute Data	1) Train d'informations qui contient les valeurs de chaque attribut que Class/Object définit pour le format de demande / réponse. Les classes/objets qui prennent en charge ce service doivent définir le format de ce paramètre. 2) Une matrice de structures, prédéfinie par le système pour le service donné
Attribute Count	Nombre de numéros d'attribut dans la liste d'attributs
Attribute list []	Liste (matrice) des numéros d'attribut des attributs à récupérer provenant de la classe ou de l'objet
Attribute Number	Numéro représentant la valeur de l'attribut
Attribute Status	Information de statut d'attribut
Attribute Value	Séquence de données spécifiques à l'attribut de l'objet ou de la classe
Object Specific Data	Paramètres spécifiques à une classe/à une instance. La spécification de la classe/de l'instance doit spécifier le format.
Instance ID	Valeur attribuée pour identifier l'objet nouvellement créé
Maximum Returned Values	Nombre maximal de valeurs d'ID d'instance à retourner dans le message de réponse
Number Of List Members	Nombre des ID d'instance spécifiés dans le message de réponse
Instance ID List	Liste des ID d'instance retournés. Les ID d'instance sont retournés dans des champs entiers de 16 bits.
IsSynchronized	Indique si l'objet est synchronisé à la Base de temps PTP

8.4 Diagrammes d'états FSPM

Le diagramme d'états FSPM n'a obtenu qu'un seul état possible: Running (En marche).

9 Diagrammes protocolaires de relation d'applications (ARPM)

9.1 Généralités

Le présent bus de terrain a des diagrammes protocolaires de relation d'applications pour:

- les relations d'applications sans connexion,
- les relations d'applications orientées connexion.

La relation sans connexion est d'un seul type uniquement.

Les relations orientées connexion s'inscrivent dans l'une de 7 classes de transport:

0 Null (or Base)
1 Duplicate Detection (Détection de doublons)
2 Acknowledged (Acquitté)
3 Verified (Vérifié)
4 Non-blocking (Non bloquant)
5 Non-blocking, Fragmenting (Non bloquant, fragmentant)
6 Multipoint, Fragmenting (Multipoint, fragmentant)

Bien que similaires, ces transports ont chacun leur propre ARPM.

9.2 ARPM sans connexion (UCMM)

9.2.1 Généralités

Les fonctions des ARPM sans connexion sont fournies par l'objet Unconnected Message Manager (UCMM, gestionnaire de messages non connectés). L'UCMM doit fournir des services de messages de demande/réponse non connectés, limités à une seule liaison qui prend en charge plusieurs messages en cours. Le nombre requis de messages en cours doit être spécifique à une mise en œuvre. L'UCMM doit être présent dans tous les nœuds et doit prendre en charge au moins un message en cours.

La spécification détaillée de l'UCMM change en fonction des différentes couches associées de liaison de données qui peuvent être associées à une couche d'application de Type 2. La couche de données sous-jacente définit la manière d'accéder à l'UCMM et peut limiter les messages que l'UCMM peut émettre.

9.2.2 Définitions de primitive

L'objet UCMM doit fournir les services suivants:

UCMM_Create	Crée une instance d'enregistrement de transaction. Place l'UCMM dans l'état "Running". Ce service est local, il n'entraîne pas l'envoi d'une PDU.
UCMM_Delete	Supprime un enregistrement de transaction existant. Place l'UCMM dans l'état "Inactive" (inactif). Ce service est local, il n'entraîne pas l'envoi d'une PDU. Ce service est local, il n'entraîne pas l'envoi d'une PDU.
UCMM_Write	Ecrit un élément de données d'application dans le tampon des PDU de transport; cela se traduit par l'envoi des données vers une cible spécifiée.
UCMM_Abort	Abandonne une transaction existante.

Pour les définitions détaillées de ces services, se référer à la CEI 61158-5-2.

Les primitives échangées entre l'UCMM et le diagramme FSPM sont montrées dans le Tableau 251 et dans le Tableau 252 ci-après.

Tableau 251 – Primitives émises par le diagramme FSPM vers le diagramme ARPM

Nom de la primitive	Source	Paramètres associés	Fonctions
UCMM_Create_req	FSPM	fixed tag max retries,	Achemine une demande du diagramme FSPM vers l'ARPM pour créer un enregistrement de transaction.
UCMM_Delete_req	FSPM	record	Achemine une demande du diagramme FSPM vers l'ARPM pour supprimer un enregistrement de transaction établi précédemment.
UCMM_Write_req	FSPM	record destination ID UCMM service response timeout transaction priority application data	Achemine une demande du diagramme FSPM vers l'ARPM pour envoyer un élément de données d'application en utilisant un enregistrement de transaction créé précédemment.
UCMM_Write_rsp	FSPM	record application data	Achemine une réponse du diagramme FSPM, en passant par un routeur de message (Message Router), vers l'ARPM pour envoyer une réponse à UCMM_Send_req en utilisant un enregistrement de transaction créé précédemment.
UCMM_Abort_req	FSPM	record	Abandonne la transaction correspondant à l'enregistrement de transaction spécifié; retire le paquet de la DLL locale s'il n'a pas encore été émis.

Tableau 252 – Primitives émises par le diagramme ARPM vers le diagramme FSPM

Nom de la primitive	Source	Paramètres associés	Fonctions
UCMM_Create_cnf	ARPM	record service_status	Achemine de l'ARPM vers le FSPM une confirmation que l'enregistrement de transaction a été créé.
UCMM_Write_ind	ARPM	record source ID application data	Achemine de l'ARPM vers le Message Router une indication que des données sont arrivées sur l'enregistrement de transaction créé précédemment. Le Message Router transmet alors l'indication à l'objet d'application approprié.
UCMM_Write_cnf	ARPM	record service status application data	Achemine de l'ARPM vers le routeur de message (Message Router) une confirmation de l'exécution de UCMM_Send_req du FSPM sur un enregistrement de transaction créé précédemment.

9.2.3 Paramètres des primitives

Les paramètres utilisés avec les primitives échangées entre le diagramme FSPM et l'ARPM sont décrits dans le Tableau 253.

Tableau 253 – Paramètres utilisés avec les primitives échangées entre le diagramme FSPM et le diagramme ARPM

Nom de paramètre	Description
record	Identifie sans ambiguïté l'instance de la connexion temporaire sur laquelle le diagramme FSPM a émis une primitive "request" UCMM_Send. Un moyen pour une telle identification n'est pas spécifié dans la présente norme.
application data	Achemine des données d'utilisateur de la FAL.
fixed tag	Mis à la valeur de 0x83 pour l'UCMM et de 0x88 pour UCMM de gestion (Management UCMM)
destination ID	Identificateur non ambigu (ID de MAC) du nœud-source de l'UCMM_write_req.
source ID	Identificateur non ambigu (ID de MAC) du nœud-destination de l'UCMM_write_req.

Nom de paramètre	Description
service	Le paramètre "service" doit spécifier le mécanisme de livraison à utiliser pour ce message et doit être l'un choisi parmi: Request_With_Acknowledge = 2 (répétitions de tentative jusqu'à l'acquittement) Request_With_No_Response = 4 (pas d'acquittement) Request_With_No_ACK = 7 (répétitions de tentative jusqu'à la réponse) Request_With_No_Retry = 8 (avec réponse) NOTE Ces valeurs correspondent à des codes de commande contenus dans l'en-tête UCMM.
timeout	Durée de la transaction en microsecondes. Si aucun UCMM_Send_cnf n'est reçu avant que la temporisation expire, la transaction est abandonnée et le paramètre send_status doit être TIMEOUT.
retries	Achemine le nombre maximal autorisé de répétitions de tentative.
create_status	Etat retourné avec UCMM_Create_conf = - success (succès) - cannot create (création impossible)
send_status	Status retourné with UCMM_Send_conf = - success (succès) - record_not_created (enregistrement non créé) - packet_too_big (paquet trop grand) - no_acknowledge (pas d'acquittement) - aborted (abandon) - timeout (temporisation)

9.2.4 Diagrammes d'états UCMM

9.2.4.1 Client UCMM

9.2.4.1.1 Etats

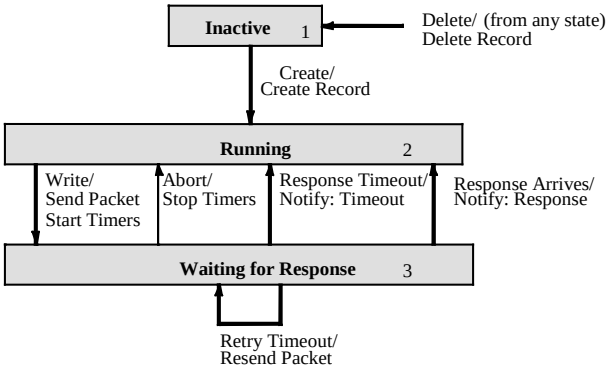
Les états définis et leurs descriptions du client UCMM sont énumérés dans le Tableau 254.

Tableau 254 – Etats du client UCMM

Etat	Description
Inactive (Inactif)	L'enregistrement pour le transfert d'UCMM n'existe pas.
Running (En marche)	L'enregistrement pour le transfert d'UCMM a été créé.
Waiting for response (En attente de réponse)	Le client a envoyé un élément de données d'application et attend une réponse.

9.2.4.1.2 Diagramme de transitions d'états

La Figure 33 montre le diagramme de transitions d'états du client UCMM.



Légende

Anglais	Français
Delete/(from any state)	Supprimer/(de n'importe quel état)
Delete Record	Supprimer l'enregistrement
Inactive	Inactif
Create/Create Record	Créer/Créer un enregistrement
Running	En marche
Write/Send Packet	Ecrire/Envoyer le paquet
Start Timers	Démarrer les temporisateurs
Abort/Stop Timers	Abandonner/Arrêter les temporisateurs
Response Timeout/Notify: Timeout	Temporisation de réponse/Notifier: temporisation
Response Arrives/Notify: Response	Réponse arrive/Notifier: temporisation
Waiting for Response	Attente d'une réponse
Retrry Timeout/Resend Packet	Temporisation de répétition de tentative/Envoyer de nouveau le paquet

Figure 33 – Diagramme de transitions d'états de client UCMM9

9.2.4.1.3 Matrice d'événements d'états

Les transitions d'états pour le client UCMM doivent être telles que spécifiées dans le Tableau 255.

Tableau 255 – Matrice d'événements d'états du client UCMM

Evénement	Etat		
	Inactive (Inactif)	Running (En marche)	Waiting for response (En attente de réponse)
Create.req	1) créer enregistrement 2) passer à Running		
Delete.req	aucune action	1) supprimer enregistrement 2) incrémenter SEQUENCE 3) passer à Inactive	1) supprimer enregistrement 2) incrémenter SEQUENCE 3) passer à Inactive
Write.req	confirmer status = INVALID_RECORD	1) envoyer paquet 2) démarrer Response Timer 3) réinitialiser le compte de répétitions de tentative 4) démarrer Retry Timer 5) passer à Waiting	confirm status = BUSY

Evénement	Etat		
	Inactive (Inactif)	Running (En marche)	Waiting for response (En attente de réponse)
Abort.req	confirmer status = INVALID_RECORD	aucune action	1) arrêter les temporisateurs 2) incrémenter SEQUENCE 3) passer à Running
Temporisation de répétitions de tentative Paquet de demande disponible			1) décrémenter le compte de répétitions de tentative SI le compte de répétitions de tentative a expiré 2) notifier: temporisation - pas d'ACK 3) libérer tampon de demandes 4) incrémenter Sequence n° 5) arrêter Response Timer 6) passer à Running SINON. 1) renvoyer paquet 2) redémarrer Retry Timer
Temporisation de réponse			1) confirmer avec statut = TIMEOUT-pas de réponse 2) incrémenter SEQUENCE 3) passer à Running
Réponse arrive		aucune action	1) confirmer avec statut = SUCCESS 2) envoyer ACK_RESP, sauf si réponse indique aucune ACK_RESP souhaitée 3) incrémenter SEQUENCE 4) passer à Running
ACK_REQ arrive, le numéro de séquence correspond à la valeur stockée.		Aucune action	1) libérer tampon de demandes 2) arrêter Retry Timer
ACK_REQ arrive, le numéro de séquence n'est pas égal à la valeur stockée.		Aucune action	Aucune action

Le numéro de séquence doit être mis à zéro à l'initialisation. La valeur du numéro de séquence doit être maintenue dans l'état "inactif", à utiliser lorsque record (l'enregistrement) passe à Running. La valeur de temporisation des répétitions de tentative doit être fixe pour la liaison à une valeur qui garantit que les nœuds tant clients que serveurs ont une opportunité d'émettre un paquet non programmé.

La temporisation de réponse est le temps de réponse fourni lorsque l'enregistrement est créé.

9.2.4.2 Serveur UCMM haut de gamme

9.2.4.2.1 Fonctions

Lorsqu'un paquet arrive au serveur, une instance de transaction doit être créée si les ressources sont disponibles. Si les ressources ne sont pas disponibles, le paquet doit être rejeté. Lorsque l'instance est créée, un enregistrement de transaction doit être créé pour l'instance en question. Cet enregistrement doit être actif pour la durée de vie de cette transaction. La transaction doit finir:

- lorsqu'un ACK_RESP est reçu après qu'une réponse a été envoyée;
- lorsque le temporisateur de temps de réponse expire; ou
- lorsqu'un nouveau paquet est reçu après qu'une réponse a été envoyée.

9.2.4.2.2 Etats

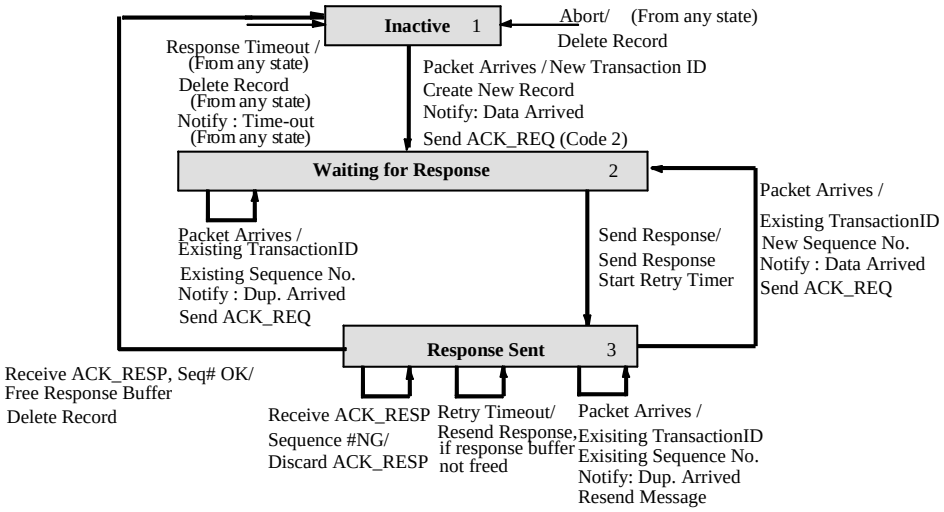
Les états définis et leurs descriptions du serveur UCMM haut de gamme sont énumérés dans le Tableau 256.

Tableau 256 – Etats du serveur UCMM haut de gamme

Etat	Description
Inactive (Inactif)	L'enregistrement pour le transfert d'UCMM n'existe pas.
Waiting for response (En attente de réponse)	Un paquet de données est arrivé avec un nouvel ID de transaction. Un nouvel enregistrement est créé. Le serveur attend que la réponse soit envoyée par l'application serveur.
Response sent (Réponse envoyée)	L'application serveur a envoyé une réponse.

9.2.4.2.3 Transitions d'états

La Figure 34 montre le diagramme de transitions d'états pour un serveur UCMM haut de gamme.



Légende

Anglais	Français
Abort/(From any state)	Abandonner/(de n'importe quel état)
Delete Record	Supprimer l'enregistrement
Inactive	Inactif
Packet Arrives/	Paquet arrive/
New Transaction ID	Nouvel ID de transaction
Create New Record	Créer nouvel enregistrement
Notify: Data Arrived	Notifier: Données arrivées
Send ACK_REQ (Code 2)	Envoyer ACK_REQ (code 2)
Response Timeout/	Temporisation de réponse
Notify: Time-out	Notifier: Temporisation
Waiting for Response	En attente de réponse
Existing Transaction ID	ID de transaction existant
New Sequence No.	Nouveau n° de séquence
Send Response/	Envoyer réponse/
Start Retry Timer	Démarrer Retry Timer
Existing Sequence No.	N° de séquence existant
Notify: Dup. Arrived	Notifier: Doublon arrivé

Anglais	Français
Response Sent	Réponse envoyée
Resend Message	Envoyer de nouveau le message
Retry Timeout/	Temporisation de répétition de tentative
Resend Response, if response buffer not freed	Envoyer de nouveau la réponse, si le tampon des réponses pas libre
Receive ACK_RESP	Recevoir ACK_RESP
Sequence #NG/	N° séquence NG/
Discard ACK_RESP	Rejeter ACK_RESP
Receive ACK_RESP, Seq# OK/	Recevoir ACK_RESP, n°Seq OK/
Free Response Buffer	Libérer le tampon des réponses

Figure 34 – Diagramme de transitions d'états d'un serveur UCMM haut de gamme

9.2.4.2.4 Matrice d'événements d'états

Les transitions d'états pour le serveur UCMM haut de gamme doivent être telles que spécifiées dans le Tableau 257.

Le numéro de séquence doit être mis à zéro à l'initialisation. La valeur du numéro de séquence doit être maintenue dans l'état "Inactive", à utiliser lorsque record (l'enregistrement) passe à Running. La valeur de temporisation des répétitions de tentative doit être fixée pour la liaison locale.

La temporisation de réponse doit être le temps de réponse fourni lorsque l'enregistrement est créé.

Tableau 257 – Matrice d'événements d'états du serveur UCMM haut de gamme

Evénement	Etat		
	Inactive (Inactif)	Waiting for response (En attente de réponse)	Response sent (Réponse envoyée)
Le paquet arrive demande (code 2) Nouvel ID de transaction et adresse source	1) Créer un nouvel enregistrement 2) Notifier: Données arrivées 3) Envoyer ACK_REQ 4) Démarrer Response Timer 5) Passer à Waiting for App Response	Non applicable	Non applicable
Le paquet arrive demande (code 7) Nouvel ID de transaction et adresse source	1) Créer un nouvel enregistrement 2) Notifier: Données arrivées 3) Démarrer Response Timer 4) Passer à Waiting for App Response	Non applicable	Non applicable
Le paquet arrive ID de transaction et adresse source existants Nouveau n° de séquence	Non applicable	Aucune action	1) Mettre à jour l'enregistrement 2) Notifier: Données arrivées 3) Envoyer ACK_REQ 4) Passer à Waiting for App Response

Evénement	Etat		
	Inactive (Inactif)	Waiting for response (En attente de réponse)	Response sent (Réponse envoyée)
Le paquet arrive ID de transaction existant N° de séquence existant	Non applicable	1) Notify: Doublon arrivé (facultatif) 2) Envoyer ACK_REQ	1) Notify: Doublon arrivé (facultatif) 2) Renvoyer message de réponse
Response Timer expire	Non applicable	1) Notify: Temporisation 2) Supprimer enregistrement 3) Passer à Inactive.	1) Notify: Temporisation 2) Supprimer enregistrement 3) Passer à Inactive.
Abandonner	Erreur	1) Supprimer enregistrement 2) Passer à Inactive.	1) Supprimer enregistrement 2) Passer à Inactive.
Envoyer réponse	Erreur	1) Envoyer réponse 2) Démarrer Retry Timer 3) Passer à Response Sent	Erreur
Temporisation des répétitions de tentative	Non applicable	Non applicable	1) Décrémenter le compte de répétitions de tentative SI le compteur de répétitions de tentative a expiré 2) Libérer tampon de réponses 3) Arrêter Response Timer 4) Passer à Inactive SINON 1) Renvoyer message de réponse 2) Démarrer Retry Timer
ACK_RESP arrive, le numéro de séquence correspond à la valeur stockée.	Aucune action	Aucune action	1) Supprimer enregistrement 2) Passer à Inactive.
ACK_RESP arrive, le numéro de séquence n'est pas égal à la valeur stockée.	Aucune action	Aucune action	Aucune action

Le temps de réponse est le temps de réponse reçu dans l'en-tête de message.

9.2.4.3 Serveur UCMM bas de gamme

9.2.4.3.1 Fonctions

Le serveur bas de gamme ne peut pas prendre en charge les fonctions suivantes:

- accomplir des répétitions de tentative de message de réponse,
- détecter les doublons de messages.

9.2.4.3.2 Etats

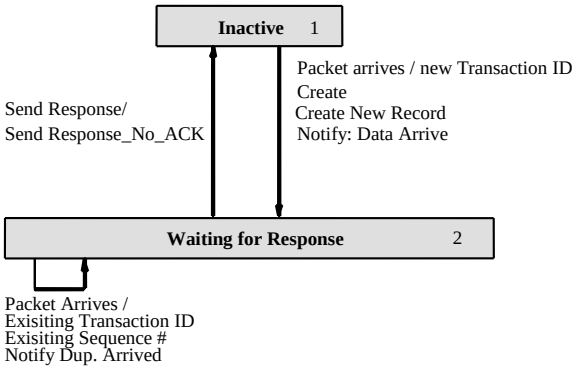
Les états définis et leurs descriptions du serveur UCMM bas de gamme sont énumérés dans le Tableau 258.

Tableau 258 – Etats du serveur UCMM bas de gamme

Etat	Description
Inactive (Inactif)	L'enregistrement pour le transfert d'UCMM n'existe pas.
Waiting for response (En attente de réponse)	Un paquet de données est arrivé avec un nouvel ID de transaction. Un nouvel enregistrement est créé. Le serveur attend que la réponse soit envoyée par l'application serveur.

9.2.4.3.3 Transitions d'états

La Figure 35 montre le diagramme de transitions d'états pour un serveur UCMM bas de gamme.



Légende

Anglais	Français
Inactive	Inactif
Packet arrives / new Transaction ID	Le paquet arrive/nouvel ID de transaction
Create	Créer
Create New Record	Créer un nouvel enregistrement
Notify: Data Arrive	Notifier: Données arrivées
Send Response/	Envoyer réponse/
Send Response_No_ACK	Envoyer Response_No_ACK
Waiting for Response	En attente de réponse
Existing Transaction ID	ID de transaction existant
Existing Sequence #	Séquence existante n°
Notify Dup. Arrived	Notifier: Doublon arrivé

Figure 35 – Diagramme de transitions d'états pour serveur UCMM bas de gamme

9.2.4.3.4 Matrice d'événements d'états

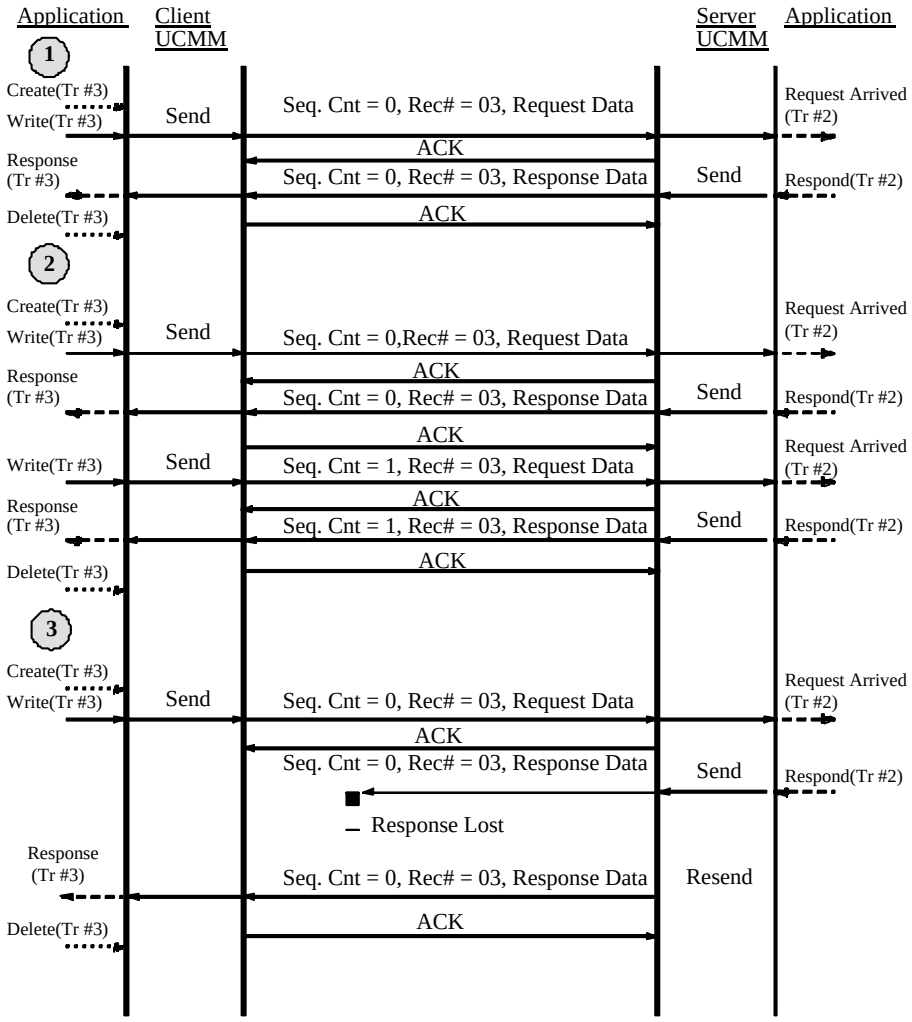
Les transitions d'états pour le serveur UCMM bas de gamme doivent être telles que spécifiées dans le Tableau 259.

Tableau 259 – Matrice d'événements d'états du serveur UCMM bas de gamme

Evénement	Etat	
	Inactive (Inactif)	Waiting for response (En attente de réponse)
Le paquet arrive Nouvel ID de transaction et nouvelle adresse source	1) Créer un nouvel enregistrement 2) Notifier: Données arrivées 3) SI réponse pas disponible immédiatement, envoyer ACK_REQ 4) Passer à Waiting for App Response	Non applicable
Le paquet arrive ID de transaction et adresse source existants Nouveau n° de séquence	Aucune action	Aucune action
Le paquet arrive ID de transaction existant N° de séquence existant	Aucune action	1) Notify: Doublon arrivé (facultatif)
Envoyer réponse.	Erreur	1) Envoyer Response_No_ACK 2) Passer à Inactive.
ACK_RESP arrive	Aucune action	Aucune action

9.2.5 Exemples de séquences UCMM

La Figure 36 montre un diagramme de séquence pour un UCMM avec un seul message en cours tandis que la Figure 37 montre un diagramme de séquence pour un UCMM avec plusieurs messages en cours.



Légende

Anglais	Français
Application	Application
Client UCMM	Client UCMM
Server UCMM	Server UCMM
Create	Créer
Write	Ecrire
Response	Répondre
Delete	Supprimer
Send	Envoyer
Request Data	Données de demande
Response Data	Données de réponse
Response Lost	Réponse perdue
Request Arrived	Demande arrivée
Respond	Répondre

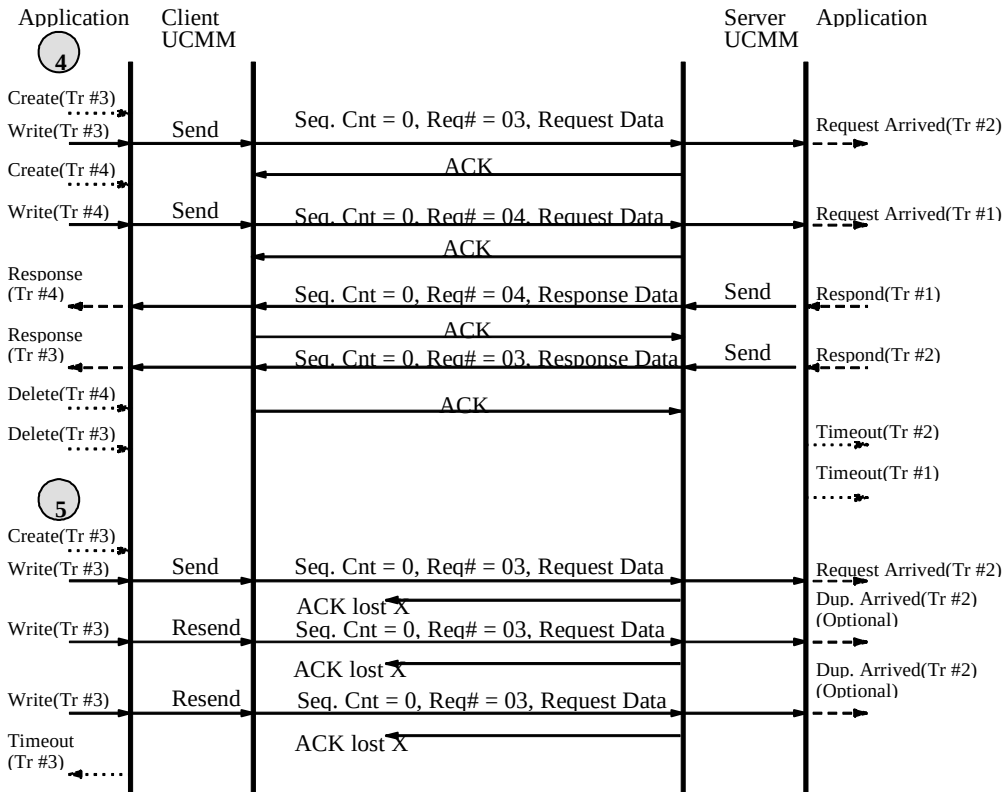
Figure 36 – Diagramme de séquence pour un UCMM avec un seul message en cours

Exemple 1. Transaction typique: Une instance est créée, la demande est écrite, la réponse est retournée et l'instance est supprimée.

Exemple 2. Deux transactions/même instance: Dans ce cas, l'instance est ouverte et deux demandes sont envoyées.

NOTE 1 Le client attend la première réponse à retourner avant d'émettre une seconde demande en utilisant le même numéro de transaction.

Exemple 3. Réponse perdue: Dans ce cas, la réponse est perdue et le service émet une répétition de tentative.



Légende

Anglais	Français
Application	Application
Client UCMM	Client UCMM
Server UCMM	Server UCMM
Create	Créer
Write	Ecrire
Response	Répondre
Delete	Supprimer
Timeout	Temporisation
Send	Envoyer
Resend	Envoyer de nouveau
Request Data	Données de demande
Response Data	Données de réponse
ACK Lost	Acquittement perdu
Request Arrived	Demande arrivée
Respond	Répondre
Dup. Arrived (Optional)	Doublon arrivé (facultatif)

Figure 37 – Diagramme de séquence pour un UCMM avec plusieurs messages en cours

Exemple 4. Plusieurs en cours: Dans ce cas, le client crée deux instances et envoie une seconde demande avant que la première demande soit achevée.

NOTE 2 Le client utilise un numéro de transaction différent pour la seconde demande.

Exemple 5. Dépassement: Dans ce cas, l'application serveur ne répond pas et le client émet une temporisation. La temporisation se traduit par le retour d'un statut d'erreur vers l'application. L'instance de transaction doit être supprimée.

9.2.6 UCMM de gestion

Tout appareil qui a mis en œuvre l'objet Keeper doit aussi mettre en œuvre le serveur Management UCMM (UCMM de gestion). A deux exceptions près, le serveur Management UCMM doit être identique au serveur UCMM à étiquette fixe 0x83 UCMM:

- les transactions du serveur Management UCMM doivent être émises sur l'étiquette fixe 0x88;
- le serveur Management UCMM ne doit émettre aucun paquet si son nœud ne contient pas un objet Keeper dans l'état de maître.

Un appareil peut mettre en œuvre le client Management UCMM. Le client Management UCMM est identique au client UCMM à étiquette fixe 0x83, excepté que les transactions pour le client Management UCMM doivent être émises sur l'étiquette fixe 0x88.

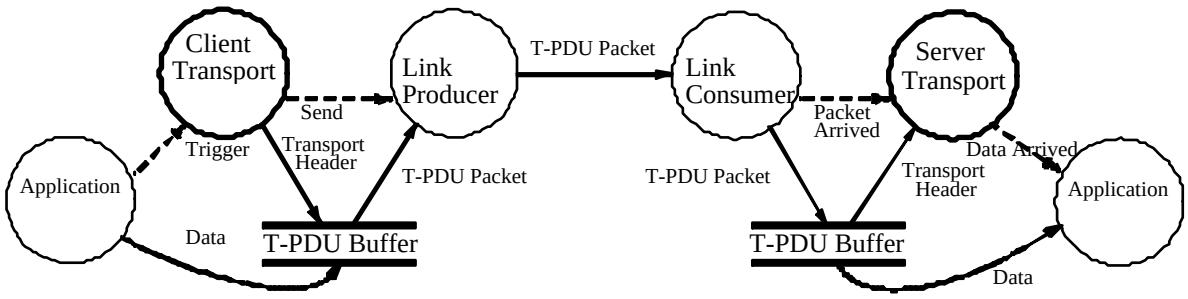
NOTE Les messages à l'objet Keeper sont diffusés, car tous les appareils sont autorisés à mettre en œuvre cet objet. Pour faciliter les communications vers l'objet Keeper sans avoir d'impact sur les appareils autres que Keeper, ces messages sont émis sur une étiquette fixe différente.

9.3 Diagrammes ARPM orientés connexion (transports)

9.3.1 Tampon des PDU de transport

9.3.1.1 Format des PDU de transport

Le tampon de TPDU contient un paquet TPDU. Le paquet TPDU est constitué d'un en-tête et de données de transport. Les applications doivent écrire les données dans les tampons TPDU et en lire des données, et ce, directement. Les instances de transports doivent écrire et lire l'en-tête de transport dans le tampon TPDU alors que les consommateurs doivent écrire les données dans le tampon TPDU et les producteurs doivent lire les données dans le tampon TPDU. Les tampons TPDU et la gestion des tampons ne font pas partie des services de communication. Ce processus est montré à la Figure 38.



Légende

Anglais	Français
Application	Application
Client Transport	Transport client
Trigger	Déclencher
Data	Données

Anglais	Français
Send	Envoyer
Transport Header	En-tête de transport
T-PDU Buffer	Tampon des T-PDU
T-PDU Packet	Paquet de T-PDU
Link Producer	Producteur de liaison
Link Consumer	Consommateur de liaison
Data Arrived	Données arrivées
Server Transport	Transport serveur
Packet arrived	Paquet arrivé

Figure 38 – Tampon des TPDU

9.3.1.2 Gestion des tampons de PDU de transport

La gestion des tampons de TPDU est spécifique à une mise en œuvre. Les mises en œuvre qui utilisent une seule mise en tampon, la double mise en tampon, l'écrasement en écriture ou la mise en file d'attente pour gérer les tampons de TPDU sont toutes autorisées.

Les réalisateurs ont la responsabilité d'assurer l'intégrité des tampons (accès atomique).

Les applications ont la responsabilité d'assurer que le tampon a été initialisé avant qu'un déclencheur ne soit émis. L'échec à initialiser le tampon doit permettre l'envoi de données inconnues. Cela est vrai pour les producteurs et les côtés client et serveur.

9.3.1.3 Notification

Les classes de transport peuvent signaler l'application par l biais d'événements. Les événements valides sont tels que spécifiés dans le Tableau 260.

Tableau 260 – Notification

Evénement	Client				Serveur			
	Classe 0	Classe 1	Classe 2	Classe 3	Classe 0	Classe 1	Classe 2	Classe 3
Données arrivées					n	n	n	n
Doublon arrivé						n	n	n
Acquittement			n					
Vérification				n				

NOTE Un événement *Doublon arrivé* ne présente normalement pas d'intérêt pour l'application; toutefois, il pourrait être utile pour déclencher le temporisateur de chien de garde afin d'indiquer que l'application client est active et assure un déclenchement.

9.3.2 Classes de transport

Les applications s'interfacent aux services de transport par le truchement des classes de transport prises en charge. Le Tableau 261 énumère les classes de transport d'usage général qui ont été définies pour les systèmes. Chaque classe de transport fournit un niveau différent de fonctionnalité. La classe de transport 0 fournit la fonctionnalité minimale requise d'une classe de transport, permettant le transfert de données entre les applications.

Tableau 261 – Classes de transport

Numéro de classe	Nom de la classe
0	Null (ou Base)
1	Duplicate Detection (Détection de doublons)
2	Acknowledged (Acquitté)
3	Verified (Vérifié)
4	Non-blocking (Non bloquant)
5	Non-blocking, Fragmenting (Non bloquant, fragmentant)
6	Multipoint, Fragmenting (Multipoint, fragmentant)

9.3.3 Définitions des primitives communes

Les classes de transport doivent fournir les services suivants:

TR_Write	Ecrit un élément de données d'application dans le tampon des PDU de transport d'envoi, prêt à être envoyé par le biais d'une connexion préétablie.
TR_Trigger	Fait envoyer par le truchement de la connexion préétablie un élément de données d'application ou de données de réponse placé au préalable dans le tampon des PDU de transport d'envoi.
TR_Packet_arrived	Indique à l'instance de transport qu'un paquet de données est arrivé dans le tampon des TPDU de transport de réception. Ce service est déclenché par une action locale.
TR_Ack_received	Indique à l'utilisateur qu'un paquet de données est arrivé dans le tampon des PDU de transport de réception signalant un acquittement de l'arrivée de données précédemment envoyées dans le tampon des PDU de transport de réception. Ce service est déclenché par une action locale.
TR_Verify	Indique à l'utilisateur qu'un paquet de données est arrivé dans le tampon des PDU de transport de réception signalant une vérification de l'arrivée de données précédemment envoyées dans l'objet d'utilisateur de réception. Ce service est déclenché par une action locale.
TR_Status_Update	Indique à l'utilisateur qu'un statut mis à jour est présent dans le tampon des PDU de transport.

Pour les définitions détaillées de ces services, se référer à la CEI 61158-5-2.

Les primitives échangées entre les instances de transport et le diagramme FSPM sont montrées dans le Tableau 262 et dans le Tableau 263 ci-après.

Tableau 262 – Primitives émises par le diagramme FSPM vers le diagramme ARPM

Nom de la primitive	Source	Paramètres associés	Fonctions
TR_Write_req	FSPM	Transport identifier Application data	Achemine une demande du diagramme FSPM vers l'ARPM pour écrire des données d'application dans le tampon des PDU de transport.
TR_Trigger.req	FSPM	Transport identifier	Achemine une demande du diagramme FSPM vers l'ARPM pour déclencher le transfert de données d'application du tampon des PDU de transport vers le Producteur de liaison.
TR_Verify.req	FSPM	Transport identifier	Achemine une demande du diagramme FSPM vers l'ARPM pour déclencher le transfert de données de vérification d'écriture vers le tampon des PDU de transport.

Tableau 263 – Primitives émises par le diagramme ARPM vers le diagramme FSPM

Nom de la primitive	Source	Paramètres associés	Fonctions
TR_Packet_arrived.ind	ARPM	Transport identifier Data arrived Duplicate arrived	Achemine une indication au FSPM qu'un élément de données est arrivé au consommateur de liaison.
TR_Ack_arrived.ind	ARPM	Transport identifier	Achemine une indication au FSPM qu'un acquittement est arrivé au consommateur de liaison.
TR_Status_Update.ind	ARPM	Transport identifier	Achemine une indication au FSPM qu'un statut nouveau est arrivé au consommateur de liaison.
TR_Verify.cnf	ARPM	Transport identifier	Achemine une confirmation de l'ARPM vers le FSPM pour indiquer la réception d'une DLPDU de vérification.

9.3.4 Paramètres des primitives communes

Les paramètres utilisés avec les primitives échangées entre la machine FSPM et la machine ARPM sont décrits dans le Tableau 264.

Tableau 264 – Paramètres utilisés avec les primitives échangées entre le diagramme FSPM et le diagramme ARPM

Nom de paramètre	Description
Transport identifier	Identificateur de l'instance du transport invoqué dans la transaction
Application data	Contient des données d'application (paramètres de demande/réponse de service ou données formatées d'utilisateur.
Data arrived Duplicate arrived	Les paramètres Data_Arrived et Duplicate_Arrived indiquent si le nouveau paquet contient de nouvelles données ou s'il est juste un doublon provenant d'un paquet reçu précédemment.

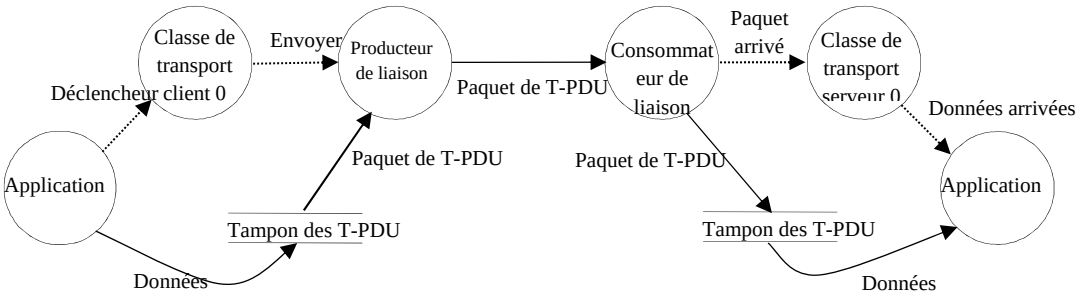
9.3.5 Diagrammes d'états de transport – classe 0

9.3.5.1 Fonctions

La classe de transport 0 est la classe de transport la plus simple et peut utiliser une connexion point à point ou multipoint. La classe 0 est typiquement utilisée pour émettre ou recevoir des entrées sur une connexion multipoint ou des sorties sur une connexion point à point.

Les utilisations possibles de la classe de transport 0 comprennent les entrées échantillonnées, les sorties normalisées, les transferts de blocs cycliques, les événements de diagnostic et la communication entre dispositifs de commande et appareils d'interface opérateur.

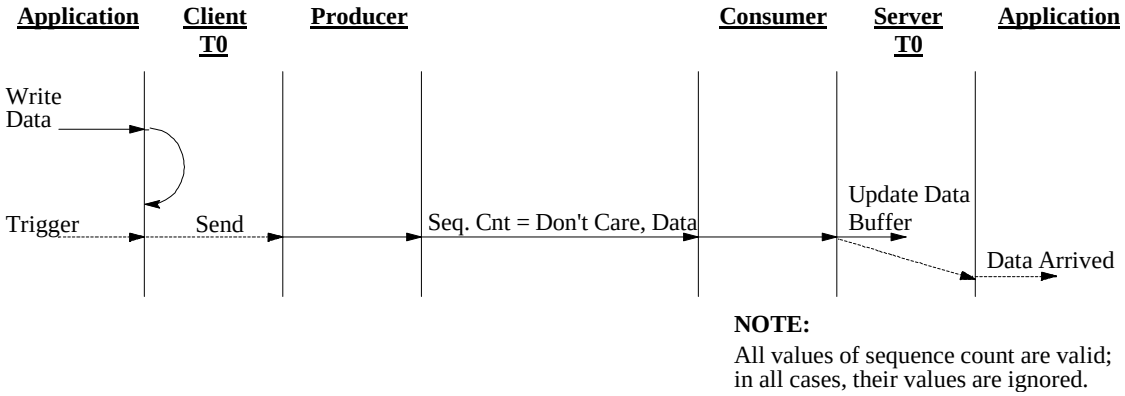
Comme la classe de transport 0 ne détecte pas les doublons de paquets de données, l'application cible a la responsabilité de les détecter. Un doublon de paquet de données est une réémission du dernier paquet de données envoyé. La Figure 39 montre les actions qui ont lieu au cours d'un transfert de données utilisant la classe de transport client 0 et la classe de transport serveur 0.



NOTE L'instance de transport client n'écrit pas un compte de séquences dans le tampon des T-PDU côté client et l'instance de transport serveur ne lit pas un compte de séquences dans le tampon des T-PDU côté serveur. L'en-tête de transport est défini comme un "don't care" (c'est-à-dire: sans influence).

Figure 39 – Diagramme de flots de données utilisant une classe de transport client 0 et une classe de transport serveur 0

La Figure 40 montre la séquence dans laquelle les actions ont lieu pendant un transfert de données utilisant la classe de transport client 0 et la classe de transport serveur 0.



Légende

Anglais	Français
Application	Application
Client T0	Client T0
Producer	Producteur
Consumer	Consommateur
Server T0	Serveur T0
Data Arrived	Données arrivées
Seq. Cnt = Don't Care, Data	Seq. Cnt = Don't Care, Data (compte de séquence = En-tête sans importance, Données
Write Data	Ecrire des données
Send	Envoyer
Trigger	Déclencher
Update Data Buffer	Mettre à jour le tampon des données
NOTE: All values of sequence count are valid; in all cases, their values are ignored.	NOTE: Toutes les valeurs du compte de séquences sont valides; dans tous les cas, leurs valeurs sont ignorées.

Figure 40 – Diagramme de séquence du transfert de données utilisant la classe de transport 0

9.3.5.2 Classe de transport client 0

9.3.5.2.1 Fonctions

La fonction de la classe de transport client 0 est juste de transmettre le service **trigger** (déclencher) d'une application vers le producteur comme un service **send** (envoyer). Il est utile que la classe de transport client 0 soit montrée du point de vue conception de haut niveau. Dans les mises en œuvre réelles, l'application peut déclencher directement le producteur. Un état de repos ("'idle") est fourni pour la cohérence avec d'autres classes de transport. Dans l'état **idle**, tous les événements, à l'exception de **delete** et de **start**, sont ignorés.

Un transport client de classe 0 a la responsabilité:

- d'accepter le déclencheur de l'application client qu'il a écrit un paquet de données dans le tampon des TPDU côté client,
- d'écrire un en-tête de transport dans le tampon de TPDU pour chaque paquet que l'application client écrit dans le tampon de TPDU côté client (facultatif),
- de déclencher le nœud producteur de la connexion réseau pour produire le paquet TPDU sur la liaison et l'envoyer vers le nœud consommateur de la connexion réseau.

9.3.5.2.2 Etats

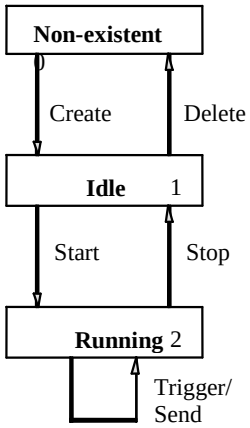
Les états définis et leurs descriptions du client de transport de classe 0 sont énumérés dans le Tableau 265.

Tableau 265 – Etats du client de transport de classe 0

Etat	Description
Non-existent (Inexistant)	L'instance de classe de transport 0 n'existe pas.
Inactif	L'instance de classe de transport 0 a été créée, mais n'a pas encore démarré.
Running (En marche)	L'instance de classe de transport 0 a été créée et a démarré.

9.3.5.2.3 Transitions d'états

Le diagramme de transitions d'états du transport client de classe 0 est montré à la Figure 41.



Légende

Anglais	Français
Create	Créer
Delete	Supprimer
Running	En marche
Trigger/Send	Déclencher/Envoyer
Start	Démarrer
Stop	Arrêter
Idle	Au repos
Non-existent	Inexistant

Figure 41 – STD client de classe 0

9.3.5.2.4 Matrice d'événements d'états

La matrice d'événements d'états du transport client de classe 0 est montrée dans le Tableau 266.

Tableau 266 – SEM client de classe 0

Evénement	Etat		
	Non-existent (Inexistant)	Inactif	Running (En marche)
Create	Passer à **Idle**	Erreur	Erreur
Delete	Erreur	Passer à **Non-existent**	Erreur
Start	Erreur	Passer à **Running**	Erreur
Stop	Erreur	Aucune action	Passer à **Idle**
Trigger	Erreur	Aucune action	Envoyer

9.3.5.3 Classe de transport serveur 0

9.3.5.3.1 Fonction

La fonction de la classe de transport serveur 0 est de transmettre l'événement **packet arrived** (paquet arrivé) du consommateur au producteur comme un événement **data arrived** (données arrivées). Il est utile que la classe de transport serveur 0 soit montrée du point de vue conception de haut niveau. Dans des mises en œuvre réelles, l'application peut recevoir l'événement directement du consommateur. Dans l'état **idle**, toutes les arrivées de paquets sont ignorées.

Un transport serveur de classe 0 a la responsabilité:

- d'accepter la notification issue du nœud consommateur de la connexion réseau qu'il a écrit un paquet de données dans le tampon des TPDU côté serveur,
- de localiser et de lire l'en-tête de transport de chaque paquet que le nœud consommateur de la connexion réseau écrit dans le tampon des TPDU côté serveur,
- de notifier à l'application serveur que des données ont été écrites dans le tampon des TPDU côté serveur.

9.3.5.3.2 Etats

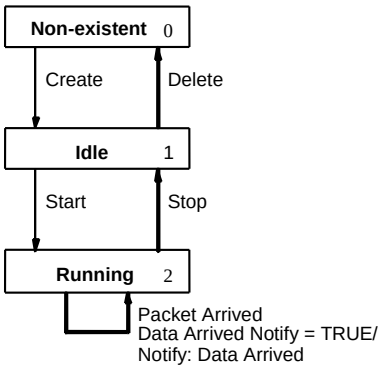
Les états définis et leurs descriptions du serveur de transport de classe 0 sont énumérés dans le Tableau 267.

Tableau 267 – Etats du serveur de transport de classe 0

Etat	Description
Non-existent (Inexistant)	L'instance de classe de transport 0 n'existe pas.
Inactif	L'instance de classe de transport 0 a été créée, mais n'a pas encore démarré.
Running (En marche)	L'instance de classe de transport 0 a été créée et a démarré.

9.3.5.3.3 Transitions d'états

Les transitions d'états du transport serveur de classe 0 sont montrées dans la Figure 42.



Légende

Anglais	Français
Create	Créer
Delete	Supprimer
Running	En marche
Start	Démarrer
Stop	Arrêter
Idle	Au repos
Non-existent	Inexistant
Packet Arrived	Paquet arrivé
Data Arrived; Notify =	Données arrivées; Notifier =
Notify: Data Arrived	Notifier: Données arrivées

Figure 42 – STD serveur de classe 0

9.3.5.3.4 Matrice d'événements d'états

La matrice d'événements d'états du transport serveur de classe 0 est montrée dans le Tableau 268.

Tableau 268 – SEM serveur de classe 0

Evénement	Etat		
	Non-existent (Inexistant)	**Inactif**	**Running (En marche)**
Create	Passer à **Idle**	Erreur	Erreur
Delete	Erreur	Passer à **Non-existent**	Erreur
Start	Erreur	Passer à **Running**	Erreur
Stop	Erreur	Aucune action	Passer à **Idle**
Packet Arrived & Data Arrived Notify = TRUE	Aucune action	Aucune action	Notification: Données arrivées

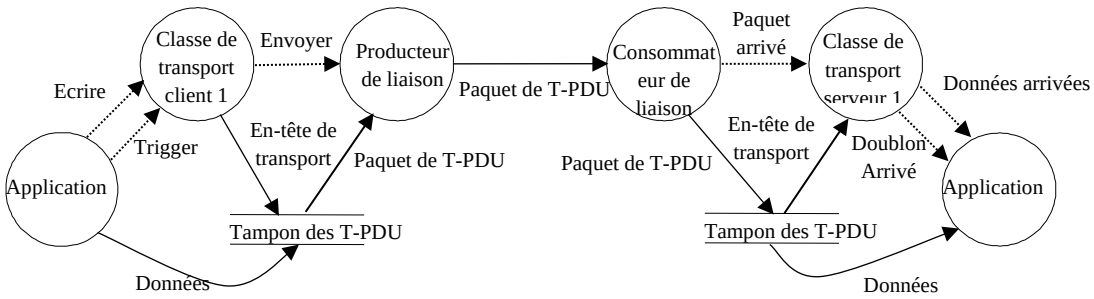
9.3.6 Diagrammes d'états de transport – classe 1

9.3.6.1 Fonctions

Comme la classe 0, la classe de transport 1 doit autoriser une connexion point à point ou multipoint. Contrairement à la classe 0, qui n'a pas d'en-tête de transport, le transport de classe 1 doit inclure un numéro de séquence, qui est utilisé pour détecter la distribution de doublons de paquets de données.

NOTE 1 La classe 1 est préférée à la classe 0 pour les cibles parce que l'application cible n'a pas à accomplir de détection de doublons, ce qui réduit le surdébit exigé dans les nœuds d'extrémité qui prennent en charge le déclencheur de transport de changement d'état. La classe 1 permet un déclencheur de changement d'état efficace, car elle a été conçue pour autoriser les transferts de données de changement d'état.

La Figure 43 montre les actions qui ont lieu au cours d'un transfert de données utilisant une instance de classe de transport client 1 et une instance de classe de transport serveur 1.



NOTE Le paquet de T-PDU comprend l'en-tête de transport issu du transport client et les données issues de l'application.

Légende

Anglais	Français
Application	Application
Client Transport Class 1	Classe de transport client 1
Trigger	Déclencher

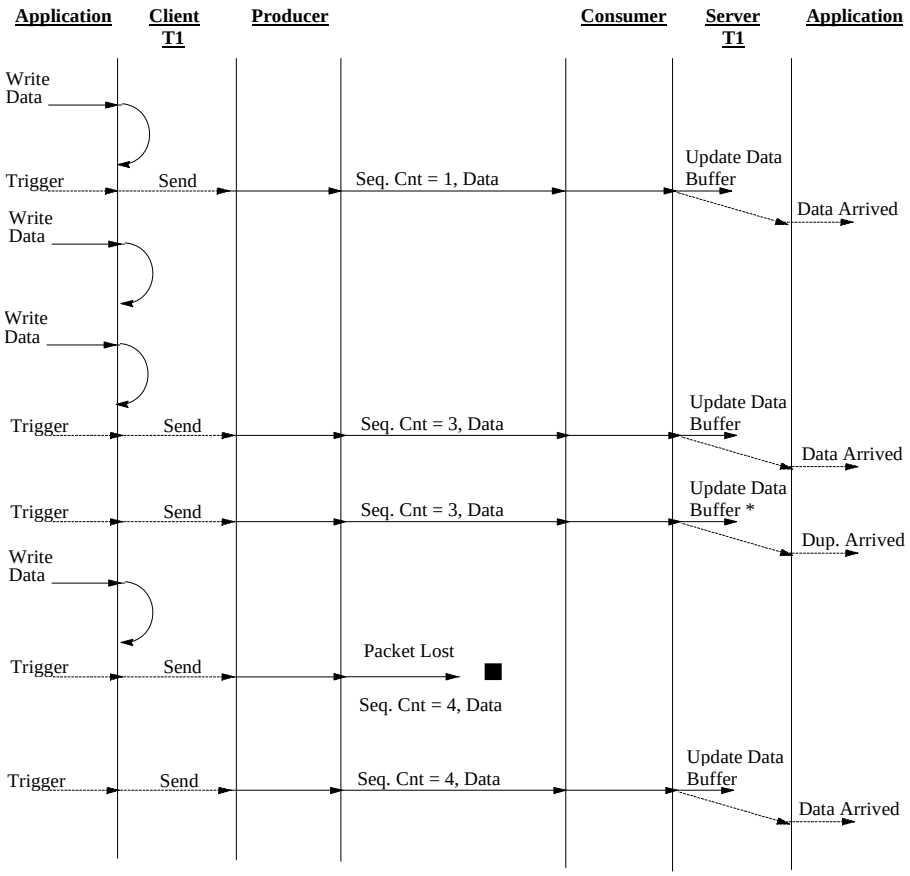
Anglais	Français
Write	Ecrire
Data	Données
Send	Envoyer
T-PDU Buffer	Tampon des T-PDU
T-PDU Packet	Paquet de T-PDU
Transport Header	En-tête de transport
Link Producer	Producteur de liaison
Link Consumer	Consommateur de liaison
Data Arrived	Données arrivées
Server Transport Class 1	Classe de transport serveur 1
Packet arrived	Paquet arrivé
Dupl. arrived	Doublon arrivé

Figure 43 – Diagramme de flots de données utilisant la classe de transport client 1 et la classe de transport serveur 1

La Figure 44 montre la séquence dans laquelle les actions ont lieu pendant un transfert de données utilisant la classe de transport client 1 et la classe de transport serveur 1.

NOTE 2 Le compte de séquences est incrémenté avec chaque écriture. En outre, lorsque le paquet est perdu sur un nouvel échantillon de données et le paquet est déclenché et envoyé ultérieurement, les données reçues par le client sont traitées comme des nouvelles données. La raison en est que c'est la première fois que le serveur a reçu ce compte de séquences. Ce mécanisme fournit une tolérance aux pannes pour les échantillons qui changent rarement.

NOTE 3 C'est un choix de mise en œuvre que de décider de mettre à jour ou non le tampon de données à la suite de la réception de doublons de données, en fonction de l'impact potentiel du traitement complémentaire.



Légende

Anglais	Français
Application	Application
Client T1	T1 client
Producer	Producteur
Consumer	Consommateur
Server T1	T1 serveur
Data	Données
Data Arrived	Données arrivées
Dup. Arrived	Doublons arrivés
Packet Lost	Paquet perdu
Send	Envoyer
Trigger	Déclencher
Update Data Buffer	Mettre à jour le tampon de données
Update Data Buffer *	Mettre à jour le tampon de données*
Write	Ecrire
Write Data	Ecrire des données

Figure 44 – Diagramme de séquence de transfert de données utilisant la classe de transport client 1 et la classe de transport serveur 1

La classe de transport 1 peut être utilisée avec un temporisateur heartbeat pour maintenir ouverte une connexion lorsqu'il existe des périodes où aucune donnée n'est émise. Le temporisateur heartbeat doit être initialisé chaque fois que le nœud producteur de la connexion réseau émet un paquet et doit être réglé de sorte que sa valeur maximale, ou la durée maximale entre des émissions de données, soit inférieure à la valeur de temporisation

du chemin pour la connexion réseau. Lorsque le temporisateur heartbeat atteint sa valeur maximale, il déclenche l'instance de transport client et, de ce fait, le nœud producteur produit et réémet le paquet dans le tampon des TPDU côté client avant que la connexion ne temporise. Le nœud consommateur de la connexion réseau, reconnaissant que le paquet a le même compte de séquences que le dernier paquet qu'il a reçu, peut ou non écraser en écriture le paquet dans le tampon des TPDU du côté serveur; cette décision de mise en œuvre est laissée à la discrétion des développeurs de produits, car ils peuvent le mieux estimer l'impact du traitement complémentaire.

Des applications pourraient utiliser la classe de transport pour distinguer entre échantillons nouveaux et échantillons anciens qui avaient été envoyés pour maintenir la connexion. Pour maintenir la connexion, un temporisateur peut être utilisé pour déclencher la production des données courantes dans le tampon des TPDU. Pour émettre de nouvelles données, l'application écrirait d'abord dans le tampon des TPDU puis déclencherait le transport client.

Des applications pourraient utiliser la classe de transport pour distinguer entre échantillons nouveaux et échantillons anciens qui avaient été envoyés pour maintenir la connexion. Pour maintenir la connexion, les événements "données arrivées" et "doublons arrivés" peuvent soumis ensemble à une opération OU logique pour réinitialiser un temporisateur. Si ce temporisateur expirait, l'application saurait qu'aucune donnée n'avait été reçue dans le délai indiqué. L'événement "données arrivées" identifierait de nouveaux échantillons de données. Ceci peut permettre à des applications de réduire leur surdébit en ne traitant que les nouveaux échantillons de données.

Les utilisations possibles de la classe de transport 1 comprennent les entrées échantillonnées, les sorties normalisées, les transferts de blocs cycliques, les événements de diagnostic et la communication entre dispositifs de commande et appareils d'interface opérateur.

9.3.6.2 Classe de transport client 1

9.3.6.2.1 Fonctions

La classe de transport 1 commence par le comportement de classe 0 et ajoute un compte de séquences. Ce compte de séquences est incrémenté par la classe de transport client lorsque des données sont écrites dans le tampon de données. Lorsque le transport reçoit un événement **write**, il doit incrémenter le compte de séquences. Lorsqu'un événement **trigger** est reçu, le transport doit mettre à jour le compte de séquences dans le tampon des TPDU et invoquer le service **send**.

Des applications pourraient utiliser la classe de transport pour distinguer entre échantillons nouveaux et échantillons anciens qui avaient été envoyés pour maintenir la connexion. Pour maintenir la connexion, un temporisateur peut être utilisé pour déclencher la production des données courantes dans le tampon des TPDU. Pour émettre de nouvelles données, l'application écrirait d'abord dans le tampon des TPDU puis déclencherait le transport client.

Un transport client de classe 1 a la responsabilité:

- d'accepter le déclencheur de l'application client qu'il a écrit un paquet de données dans le tampon des TPDU côté client,
- d'écrire un en-tête de transport dans le tampon des TPDU pour chaque paquet que l'application client écrit dans le tampon des TPDU côté client,
- d'initialiser le compte de séquences dans l'en-tête de transport (pour le premier paquet émis) ou d'incrémenter le compte de séquences (pour les paquets ultérieurs de la même émission),
- de déclencher le nœud producteur de la connexion réseau pour produire le paquet TPDU sur la liaison et l'envoyer vers le nœud consommateur de la connexion réseau.

9.3.6.2.2 Etats

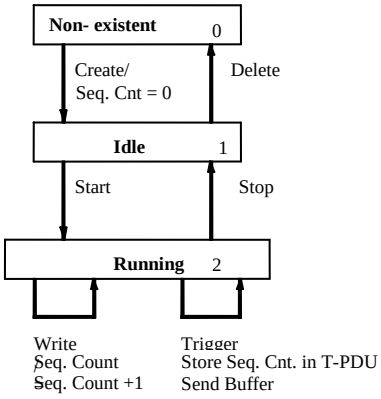
Les états définis et leurs descriptions du client de transport de classe 1 sont énumérés dans le Tableau 269.

Tableau 269 – Etats du client de transport de classe 1

Etat	Description
Non-existent (Inexistant)	L'instance de classe de transport 1 n'existe pas.
Inactif	L'instance de classe de transport 1 a été créée, mais n'a pas encore démarré.
Running (En marche)	L'instance de classe de transport 1 a été créée et a démarré.

9.3.6.2.3 Transitions d'états

Les transitions d'états du transport client de classe 1 sont montrées à la Figure 45.



NOTE Le compte de séquences est mis à 0 par le service create. Il est ensuite incrémenté après chaque événement *write* tant qu'il n'a pas atteint une valeur maximale de 2^{16}-1. Après quoi, il repasse par 0.

Légende

Anglais	Français
Non-existent	Inexistant
Create Seq. count =0	Créer Compte de séq. = 0
Delete	Supprimer
Idle	Au repos
Start	Démarrer
Stop	Arrêter
Running	En marche
Write	Ecrire
Trigger	Déclencher
Store Seq. Cnt. In T-PDU	Stocker compte de séq. dans T-PDU
Send Buffer	Envoyer tampon

Figure 45 – STD client de classe 1

9.3.6.2.4 Matrice d'événements d'états

Les transitions d'états du transport client de classe 1 sont montrées au Tableau 270.

Tableau 270 – SEM client de classe 1

Evénement	Etat		
	Non-existent (Inexistant)	**Inactif**	**Running (En marche)**
Create	1) Passer à **Idle** 2) Mettre seq. count = 0	Erreur	Erreur
Delete	Erreur	Passer à **Non-existent**	Erreur
Start	Erreur	Passer à **Running**	Erreur
Stop	Erreur	Aucune action	Passer à **Idle**
Ecrire	Erreur	Aucune action	1) Seq. Cnt = Seq. Cnt + 1
Trigger	Erreur	Aucune action	1) Stocker Seq. Cnt. dans le tampon des TPDU 2) Envoyer

9.3.6.3 Classe de transport serveur 1

9.3.6.3.1 Fonctions

La classe de transport 1 commence par le comportement de classe 0 et ajoute un compte de séquences. Le compte de séquences reçu est comparé au précédent compte de séquences. S'ils sont égaux, le paquet reçu est considéré comme étant un doublon de paquet. S'ils ne sont pas égaux, les données reçues sont considérées comme étant un nouvel échantillon. Il n'est effectué aucune tentative de compter combien de comptes de séquences ont changé entre les échantillons.

Des applications pourraient utiliser la classe de transport pour distinguer entre échantillons nouveaux et échantillons anciens qui avaient été envoyés pour maintenir la connexion. Pour maintenir la connexion, les événements "données arrivées" et "doublons arrivés" peuvent soumis ensemble à une opération OU logique pour réinitialiser un temporisateur. Si ce temporisateur expirait, l'application saurait qu'aucune donnée n'avait été reçue dans le délai indiqué. L'événement "données arrivées" identifierait de nouveaux échantillons de données. Ceci peut permettre à des applications de réduire leur surdébit en ne traitant que les nouveaux échantillons de données.

Un transport serveur de classe 1 a la responsabilité:

- d'accepter la notification issue du nœud consommateur de la connexion réseau qu'il a écrit un paquet de données dans le tampon des TPDU côté serveur,
- de localiser et de lire l'en-tête de transport de chaque paquet que le nœud consommateur de la connexion réseau écrit dans le tampon des TPDU côté serveur,
- de déclencher l'application serveur que des données ont été écrites dans le tampon des TPDU côté serveur.
- de comparer le compte de séquences de chaque paquet avec celui du paquet précédent,
- de notifier à l'application serveur que le paquet arrivé le plus récemment est un doublon du précédent lorsque leurs comptes de séquences concordent.

9.3.6.3.2 Etats

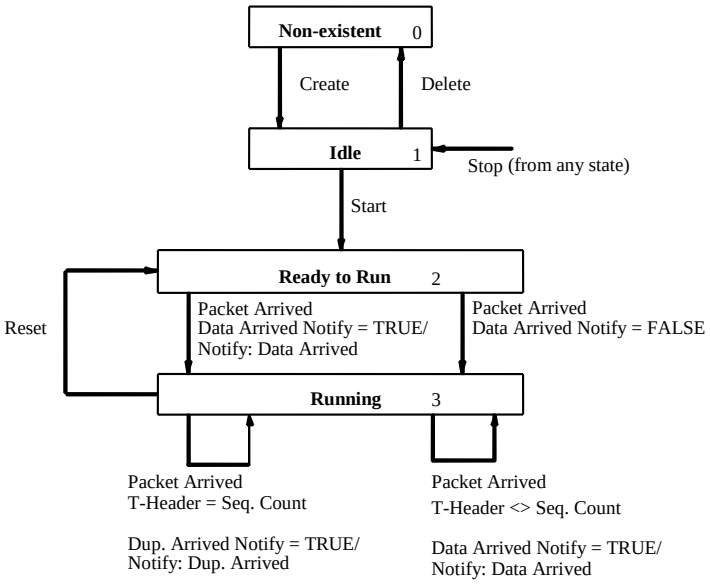
Les états définis et leurs descriptions du serveur de transport de classe 1 sont énumérés dans le Tableau 271.

Tableau 271 – Etats du serveur de transport de classe 1

Etat	Description
Non-existent (Inexistant)	L'instance de classe de transport 1 n'existe pas.
Inactif	L'instance de classe de transport 1 a été créée, mais n'a pas encore démarré.
Ready to run (prêt à fonctionner)	L'instance de la classe de transport 1 a été créée et a démarré, attendant qu'un premier paquet de données arrive.
Running (En marche)	L'instance de la classe de transport 1 a été créée et a démarré, le premier paquet de données est arrivé.

9.3.6.3.3 Transitions d'états

Les transitions d'états du transport serveur de classe 1 sont montrées dans la Figure 46.



Légende

Anglais	Français
Non-existent	Inexistant
Create	Créer
Delete	Supprimer
Idle	Au repos
Stop (from any state)	Arrêter (de n'importe quel état)
Start	Démarrer
Ready to Run	Prêt à fonctionner
Packet Arrived Data Arrived Notify = FALSE	Paquet arrivé Notification de données arrivées = FALSE
Packet Arrived Data Arrived Notify = TRUE/ Notify: Data Arrived	Paquet arrivé Notification de données arrivées = TRUE/ Notifier: Données arrivées
Reset	Réinitialiser
Running	En marche

Anglais	Français
Packet Arrived T-Header <> Seq. Count	Paquet arrivé En-tête Transport <> Compte séquences
Data Arrived Notify = TRUE/ Notify: Data Arrived	Notification de données arrivées = TRUE/ Notifier: Données arrivées
Packet Arrived T-Header = Seq. Count	Paquet arrivé En-tête Transport = Compte séquences
Dup. Arrived Notify = TRUE/ Notify: Dup. Arrived	Notification de doublons arrivés = TRUE/ Notifier: Doublons arrivées

Figure 46 – STD serveur de classe 1

9.3.6.3.4 Matrice d'événements d'états

Les transitions d'états du transport serveur de classe 1 sont montrées dans le Tableau 272.

Tableau 272 – SEM serveur de classe 1

Evénement	Etat/			
	Non-existent (Inexistant)	**Inactif**	**Ready to run (prêt à fonctionner)**	**Running (En marche)**
Create	Passer à **Idle**	Erreur	Erreur	Erreur
Delete	Erreur	Passer à **Non-existent**	Erreur	Erreur
Start	Erreur	Passer à **Ready to Run**	Erreur	Erreur
Stop	Erreur	Aucune action	Passer à **Idle**	Passer à **Idle**
Reset	Erreur	Erreur	Aucune action	Passer à **Ready to Run**
Paquet arrivé En-tête de transport <> Seq. Count Données arrivées Notify = TRUE Doublon arrivé Notify = TRUE	Aucune action	Aucune action	Passer à **Running** Notifier : Données arrivées	Notification: Données arrivées
Paquet arrivé En-tête de transport <> Seq. Count Données arrivées Notify = TRUE Doublon arrivé Notify = TRUE	Aucune action	Aucune action	Passer à **Running** Notifier: Données arrivées	Notification: Doublon arrivé
Paquet arrivé En-tête de transport <> Seq. Count Données arrivées Notify = TRUE Doublon arrivé Notify = FALSE	Aucune action	Aucune action	Passer à **Running** Notifier: Données arrivées	Notification: Données arrivées
Paquet arrivé En-tête de transport <> Seq. Count Données arrivées Notify = TRUE Doublon arrivé Notify = FALSE	Aucune action	Aucune action	Passer à **Running** Notifier: Données arrivées	Aucune action
Paquet arrivé En-tête de transport <> Seq. Count Données arrivées Notify = FALSE Doublon arrivé Notify = TRUE	Aucune action	Aucune action	Passer à **Running**	Aucune action

Evénement	Etat/			
	Non-existent (Inexistant)	Inactif	Ready to run (prêt à fonctionner)	Running (En marche)
Paquet arrivé En-tête de transport <> Seq. Count Données arrivées Notify = FALSE Doublon arrivé Notify = TRUE	Aucune action	Aucune action	Passer à **Running**	Notification: Doublon arrivé
Paquet arrivé En-tête de transport <> Seq. Count Données arrivées Notify = FALSE Doublon arrivé Notify = FALSE	Aucune action	Aucune action	Passer à **Running**	Aucune action
Paquet arrivé En-tête de transport <> Seq. Count Données arrivées Notify = FALSE Doublon arrivé Notify = FALSE	Aucune action	Aucune action	Passer à **Running**	Aucune action

9.3.7 Diagrammes d'états de transport – classe 2

9.3.7.1 Fonctions

La classe de transport 2 commence par le comportement de classe 1 et ajoute une connexion de retour qui acquitte la livraison d'un paquet. La connexion de producteur à partir du nœud client peut être point à point ou multipoint. La connexion du serveur vers le client est point à point.

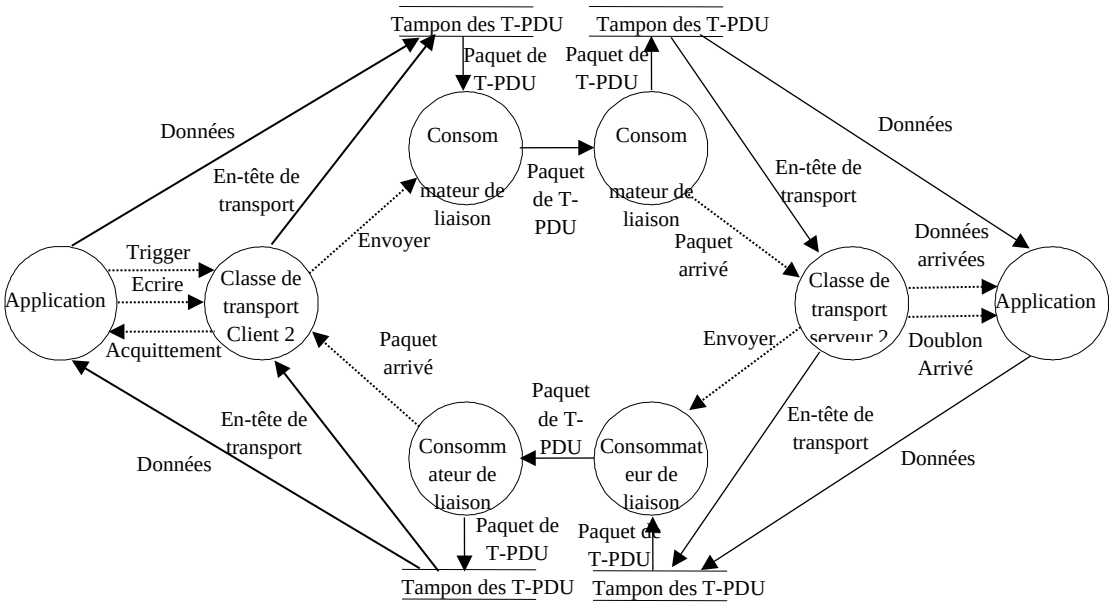
Cette classe de transport permet de notifier à l'application client que le serveur a reçu le paquet émis. Cet acquittement dit seulement à l'application client que les données sont arrivées. Cela ne signifie pas que l'application serveur ait lu le tampon ou qu'un nouvel échantillon puisse être envoyé sans écraser en écriture le tampon.

Le principal avantage de cette classe sur la classe 3 (Verified) est que l'acquittement est retourné immédiatement après que le paquet arrive au consommateur, alors que la classe de vérification exige que l'application déclenche la vérification de retour. Les retards associés à la réception de l'acquittement sont faibles, approximativement deux fois le temps requis pour une émission dans un seul sens.

La classe de transport 2 utilise deux connexions réseau: la connexion dans le sens "émetteur vers cible", qui peut être point à point ou multipoint; et la connexion de retour, qui doit être point à point. Une connexion multipoint en utilisant la classe de transport 2 doit avoir une connexion de retour point à point correspondant à chaque connexion dans le sens "client vers serveur". L'acquittement notifie à l'instance de transport client que le nœud consommateur de la connexion réseau a reçu le paquet.

Si une instance de transport serveur n'acquitte pas la réception du paquet, l'instance de transport client déclenche l'application client pour réémettre le paquet envoyé le plus récemment; si la connexion client vers serveur est l'une des connexions réseau d'une connexion multipoint, les données sont renvoyées sur toutes les connexions réseau de la connexion multipoint. L'application serveur peut retourner des données à l'application client accompagnées de son acquittement.

La Figure 47 montre les actions qui ont lieu au cours d'un transfert de données utilisant la classe de transport client 2 et la classe de transport serveur 2.



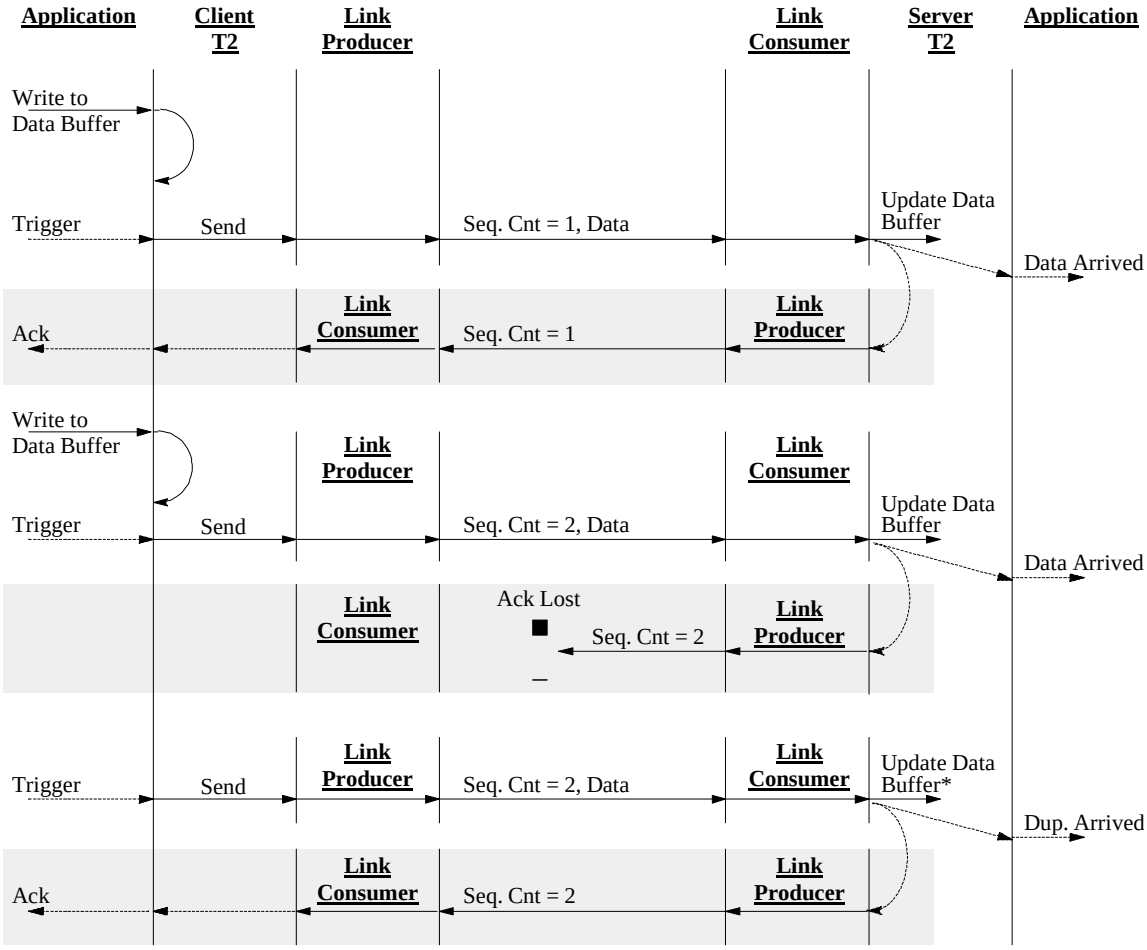
NOTE 1 Les transports client et serveur utilisent chacun deux tampons: un pour la transmission, un autre pour la réception.

NOTE 2 L'application cible n'est pas impliquée dans l'envoi de l'acquittement.

Figure 47 – Diagramme de flots de données utilisant la classe de transport client 2 et la classe de transport serveur 2

L'instance de transport serveur déclenche et envoie l'acquittement à l'instance de transport client pour notifier à l'application client qu'il peut écrire le prochain paquet de données dans le tampon d'émission des TPDU côté client. Chaque acquittement doit être reçu par l'instance de transport client avant que le prochain paquet ne soit envoyé sur la(les) même(s) connexion(s) réseau dans le sens client vers serveur.

La Figure 48 montre les actions qui ont lieu au cours d'un transfert de données utilisant la classe de transport client 2 et la classe de transport serveur 2. Dans l'exemple décrit, l'acquittement est la seule donnée transmise du serveur au client. Les zones non ombrées de la Figure 48 indiquent une émission de données dans le sens client vers serveur tandis que les zones ombrées indiquent une émission dans le sens serveur vers client.



* Implementors must decide whether to update the data buffer upon receipt of duplicate data packets, since they can best assess the impact of the additional processing.

Légende

Anglais	Français
Application	Application
ClientT2	ClientT2
Link Producer	Producteur de liaison
Link Consumer	Consommateur de liaison
Server T2	Serveur T2
Write to Data Buffer	Ecrire dans le tampon de données
Trigger	Déclencher
Send	Envoyer
Seq. Cnt = 1, Data	Compte de séquence = 1, Données
Seq. Cnt = 2	Compte de séquence = 2
Update Data Buffer	Mettre à jour le tampon de données
Data Arrived	Données arrivées
Ack	Acquittement
Ack Lost	Acquittement perdu
Update Data Buffer*	Mettre à jour le tampon de données*
* Implementors must decide whether to update the data buffer upon receipt of duplicate data packets since they can best assess the impact of the additional processing.	* Les réalisateurs doivent décider de mettre à jour ou pas le tampon de données à la réception de doublons de paquets de données, car ils peuvent le mieux estimer l'impact du traitement complémentaire.

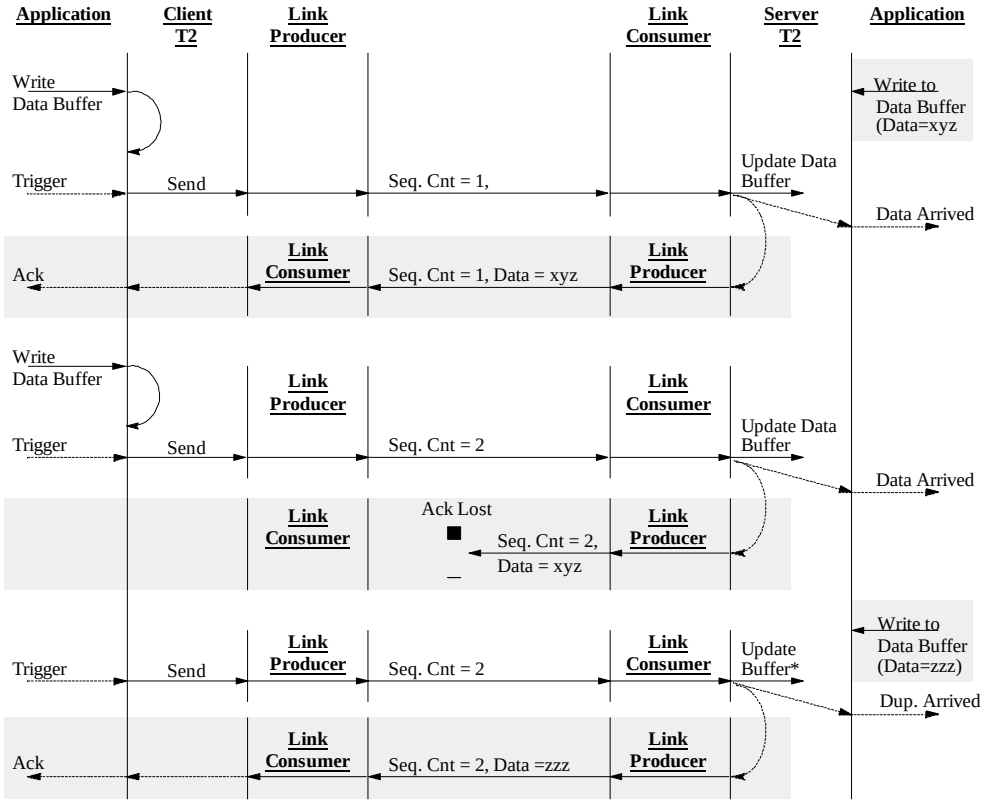
Figure 48 – Diagramme de transfert de données utilisant la classe de transport client 2 et la classe de transport serveur 2 sans données retournées

Par moments, l'émission dans le sens serveur vers client comprend l'acquittement et des données complémentaires.

La Figure 49 montre la séquence dans laquelle des actions ont lieu au cours du transfert de données utilisant la classe de transport client 2 et la classe de transport serveur 2 lorsque l'acquittement et des données sont retournés au client.

NOTE Les données retournées sont écrites de façon asynchrone dans l'accusé de réception des données issu du client.

Les zones non ombrées de la Figure 49 indiquent une émission de données dans le sens client vers serveur tandis que les zones ombrées indiquent une émission de données dans le sens serveur vers client.



* Implementors must decide whether to update the data buffer upon receipt of duplicate data packets, since they can best assess the impact of the additional processing.

Légende

Anglais	Français
Application	Application
ClientT2	ClientT2
Link Producer	Producteur de liaison
Link Consumer	Consommateur de liaison
Server T2	Serveur T2
Write to Data Buffer	Ecrire dans le tampon de données
Trigger	Déclencher
Send	Envoyer
Seq. Cnt = 1, Data	Compte de séquence = 1, Données
Seq. Cnt = 2	Compte de séquence = 2
Update Data Buffer	Mettre à jour le tampon de données
Data Arrived	Données arrivées
Ack	Acquittement

Anglais	Français
Ack Lost	Acquittement perdu
Update Data Buffer*	Mettre à jour le tampon de données*
* Implementors must decide whether to update the data buffer upon receipt of duplicate data packets since they can best assess the impact of the additional processing.	* Les réalisateurs doivent décider de mettre à jour ou pas le tampon de données à la réception de doublons de paquets de données, car ils peuvent le mieux estimer l'impact du traitement complémentaire.

Figure 49 – Diagramme de séquence de transfert de données utilisant la classe de transport client 2 et la classe de transport serveur 2 sans données retournées

Les utilisations possibles de la classe de transport 2 comprennent les applications maître/esclave et la communication entre des dispositifs de commande et des appareils d'interface opérateur.

9.3.7.2 Classe de transport client 2

9.3.7.2.1 Fonctions

Dans le sens client vers serveur, un transport client de classe *2 a la responsabilité:

- d'accepter le déclencheur de l'application client qu'il a écrit un paquet de données dans le tampon d'émission des TPDU côté client;
- d'écrire un en-tête de transport au tampon de transmission TPDU côté client pour chaque paquet que lui écrit l'application client; l'en-tête de transport pour chaque paquet doit comprendre un compte de séquence;
- d'initialiser le compte de séquences dans l'en-tête de transport (pour le premier paquet émis) ou d'incrémenter le compte de séquences (pour les paquets ultérieurs de la même émission);
- de déclencher le nœud producteur de la connexion réseau pour produire le paquet TPDU sur la liaison et l'envoyer vers le nœud consommateur de la connexion réseau.

Dans le sens serveur vers client, un transport client de classe 2 a la responsabilité:

- d'accepter la notification issue du nœud consommateur de la connexion réseau de retour qu'il a écrit des données (l'acquittement de l'instance de transport serveur) dans le tampon de réception des TPU côté client;
- de localiser et de lire l'en-tête de transport de chaque paquet de données que le nœud consommateur de la connexion réseau de retour écrit dans le tampon de réception des TPDU côté client;
- de notifier à l'application client que le nœud consommateur de la connexion réseau de retour a écrit des données dans le tampon de réception des TPDU côté client.

L'application peut mettre à jour le tampon de données ou redéclencher les données avant que tous les acquittements n'aient été reçus. Si le transport client est redéclenché alors qu'il attend un acquittement, le registre d'acquittement précédent reçu est effacé. Cela signifie que la classe de transport client doit attendre que tous les consommateurs actifs retournent un acquittement avant de faire notification à l'application.

9.3.7.2.2 Etats

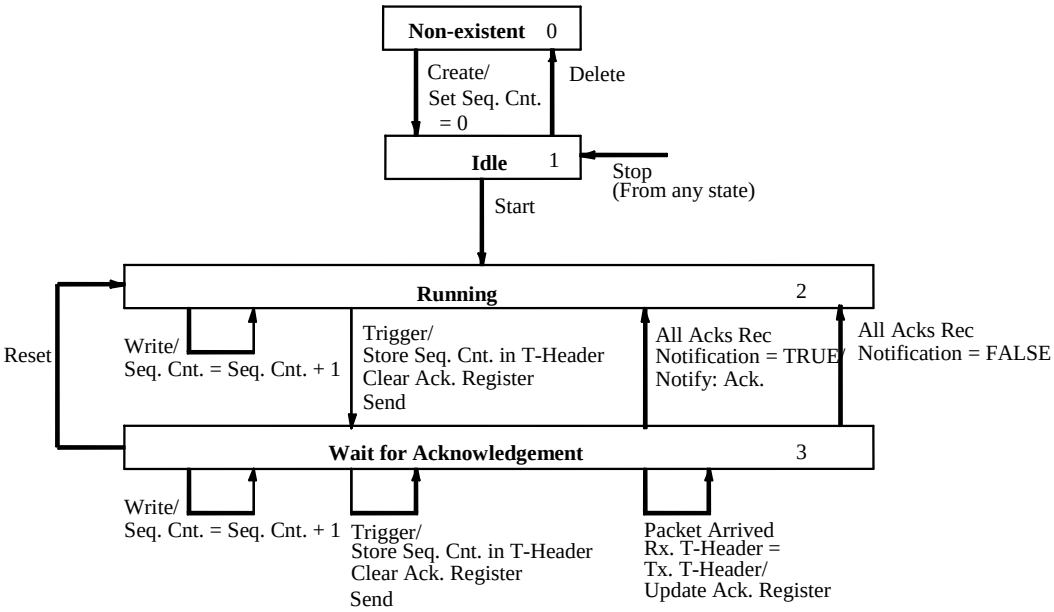
Les états définis et leurs descriptions du client de transport de classe 2 sont énumérés dans le Tableau 273.

Tableau 273 – Etats du client de transport de classe 2

Etat	Description
Non-existent (Inexistant)	L'instance de classe de transport 2 n'existe pas.
Inactif	L'instance de classe de transport 2 a été créée, mais n'a pas encore démarré.
Running (En marche)	L'instance de classe de transport 2 a été créée et a démarré.
Wait for acknowledgement (Attendre un acquittement)	L'instance de la classe de transport 2 a été créée et a démarré, l'acquittement est en cours.

9.3.7.2.3 Transitions d'états

Les transitions d'états du transport client de classe 2 sont montrées à la Figure 50.



NOTE Le compte de séquences est mis à 0 par le service create. Il est ensuite incrémenté après chaque événement *write* tant qu'il n'a pas atteint une valeur maximale de 2^{16}-1. Après quoi, il repasse par 0.

Légende

Anglais	Français
Non-existent	Inexistant
Create/ Set Seq. Cnt. = 0	Créer/ Mettre compte séq = 0
Delete	Supprimer
Idle	Au repos
Start	Démarrer
Stop (From any state)	Arrêter (de n'importe quel état)
Running	En marche
All Acks Rec	Tous les acquittements reçus
Notification = FALSE	Notification = FALSE
All Acks Rec	Tous les acquittements reçus
Notification = TRUE/	Notification = TRUE/
Notify: Ack.	Notification: acquittement

Anglais	Français
Trigger/	Déclencher/
Store Seq. Cnt. in T-Header	Stocker le cmpte séq dans l'en-tête T
Clear Ack. Register	Effacer le registre des acquitt.
Send	Envoyer
Write/	Ecrire/
Seq. Cnt. = Seq. Cnt. + 1	Seq. Cnt. = Seq. Cnt. + 1
Reset	Réinitialiser
Wait for Acknowledgement	Attendre un acquittement
Write/	Ecrire/
Seq. Cnt. = Seq. Cnt. + 1	Seq. Cnt. = Seq. Cnt. + 1
Trigger/ Store Seq. Cnt. in T-Header	Déclencher/ Stocker le cmpte séq dans l'en-tête T
Clear Ack. Register	Effacer le registre des acquitt.
Packet Arrived	Paquet arrivé
Rx. T-Header =	En-tête T, réception=
Tx. T-Header/	En-tête T, émission/

Figure 50 – STD client de classe 2

9.3.7.2.4 Matrice d'événements d'états

Les transitions d'états du transport client de classe 2 sont montrées au Tableau 274.

Tableau 274 – SEM client de classe 2

Evénement	Etat			
	Non-existent (Inexistant)	**Inactif**	**Running (En marche)**	**Wait for ack**
Create	1) Seq. Cnt = 0 2) Passer à **Idle**	Erreur	Erreur	Erreur
Delete	Erreur	Passer à **Non-existent**	Erreur	Erreur
Start	Erreur	Passer à **Running**	Erreur	Erreur
Stop	Erreur	Aucune action	Passer à **Idle**	Passer à **Idle**
Reset	Erreur	Erreur	Aucune action	Passer à **Running**
Ecrire	Erreur	Aucune action	1) Seq. Cnt = Seq. Cnt + 1 2) Ecrire TPDU	1) Seq.Cnt. = Seq.Cnt. +1 2) Ecrire TPDU
Trigger	Erreur	Aucune action	1) Stocker Seq. Cnt. dans la TPDU 2) Effacer le registre d'Ack. 3) Envoyer 4) Passer à **Wait for Ack.**	1) Stocker Seq. Cnt. dans la TPDU 2) Effacer le registre d'Ack. 3) Envoyer
Paquet arrivé En-tête T Rx. = En-tête T Rx.	Erreur	Aucune action	Aucune action	1) Mettre à jour registre des Ack. Register
Tous les Ack. reçus Ack. Notify = TRUE	Erreur	Aucune action	Erreur	1) Notify: Acquittement 2) Passer à **Running**
Tous les Ack. reçus Ack. Notify = FALSE	Erreur	Aucune action	Erreur	1) Passer à **Running**

NOTE Le STD et la SEM pour les classes client 2 et 3 sont presque identiques. "Verify" (vérification) a remplacé "acknowledgement" (acquittement).

9.3.7.3 Classe de transport serveur 2

9.3.7.3.1 Fonctions

La classe de transport serveur 2 commence par le comportement de la classe de transport 1 et ajoute une connexion dans le sens serveur vers client qui est utilisée pour acquitter un message. Cette connexion de retour est utilisée pour identifier le message d'acquittement.

Après avoir reçu un événement **packet arrived**, le transport serveur doit lire l'en-tête de transport reçu, le stocker dans l'en-tête de transport d'émission et invoquer le service **send**. L'application n'est pas impliquée dans l'acquittement. La connexion de producteur à partir du nœud client peut être point à point ou multipoint. La connexion du serveur vers le client est point à point.

Dans le sens client vers serveur, un transport serveur de classe 2 a la responsabilité:

- d'accepter la notification issue du nœud consommateur de la connexion réseau qu'il a écrit un paquet de données dans le tampon de réception des TPDU côté serveur;
- de localiser et de lire l'en-tête de transport de chaque paquet que le nœud consommateur de la connexion réseau écrit dans le tampon de réception des TPDU côté serveur;
- de comparer le compte de séquences de chaque paquet dans le tampon de réception des TPDU côté serveur à celui du paquet précédent;
- de notifier à l'application serveur que le paquet arrivé le plus récemment est un doublon du précédent lorsque leurs comptes de séquences concordent;
- de notifier à l'application serveur que des données ont été écrites dans le tampon de réception des TPDU côté serveur. (ACK peut contenir de données serveur.)

Dans le sens serveur vers client, un transport serveur de classe 2 a la responsabilité:

- d'écrire dans le tampon d'émission des TPDU côté serveur l'en-tête de transport (compte de séquences compris) des données dans le tampon de réception des TPDU côté serveur;
- de déclencher le nœud producteur de la connexion réseau de retour pour produire le paquet TPDU sur la liaison et l'envoyer vers le nœud consommateur de la connexion réseau de retour.

9.3.7.3.2 Etats

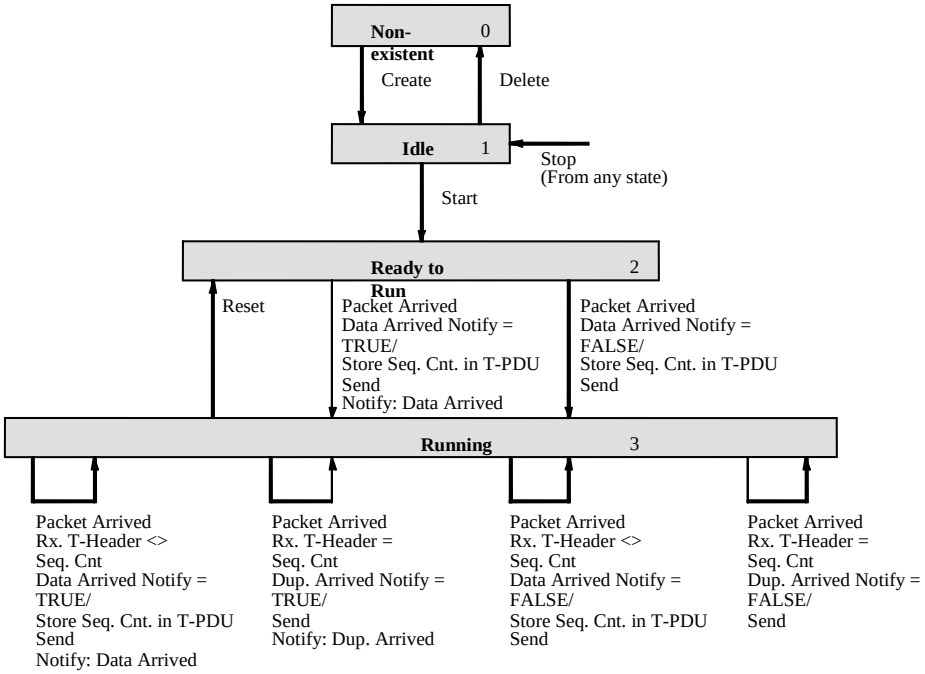
Les états définis et leurs descriptions du serveur de transport de classe 2 sont énumérés dans le Tableau 275.

Tableau 275 – Etats du serveur de transport de classe 2

Etat	Description
Non-existent (Inexistant)	L'instance de classe de transport 2 n'existe pas.
Inactif	L'instance de classe de transport 2 a été créée, mais n'a pas encore démarré.
Running (En marche)	L'instance de classe de transport 2 a été créée et a démarré.

9.3.7.3.3 Transitions d'états

Les transitions d'états du transport serveur de classe 2 sont montrées dans la Figure 51.



Légende

Anglais	Français
Nonexistent	Inexistant
Create	Créer
Delete	Supprimer
Idle	Au repos
Stop (From any state)	Arrêter (De n'importe quel état)
Start	Démarrer
Ready to Run	Prêt à fonctionner
Packet Arrived Data Arrived Notify =FALSE/ Store Seq. Cnt. in T-PDU Send	Paquet arrivé Notification de données arrivées = FALSE/ Stocker le cmpte séqn dans T-PDU Envoyer
Packet Arrived Data Arrived Notify =TRUE/ Store Seq. Cnt. in T-PDU Send Notify: Data Arrived	Paquet arrivé Notification de données arrivées = TRUE/ Stocker le cmpte séqn dans T-PDU Envoyer Notifier: données arrivées
Reset	Réinitialiser
Running	En marche
Packet Arrived Rx. T-Header <>Seq. Cnt Data Arrived Notify =TRUE/ Store Seq. Cnt. in T-PDU Send Notify: Data Arrived	Paquet arrivé Rx. T-Header <>Seq. Cnt Notification de données arrivées = TRUE/ Stocker le cmpte séqn dans T-PDU Envoyer Notifier: données arrivées

Anglais	Français
Packet Arrived Rx. T-Header =Seq. Cnt Dup. Arrived Notify =TRUE/ Send Notify: Dup. Arrived	Paquet arrivé Rx. T-Header = Seq. Cnt Notification de doublons arrivé = TRUE/ Stocker le cmpte séqn dans T-PDU Envoyer Notifier: Doublons arrivés
Packet Arrived Rx. T-Header <>Seq. Cnt Data. Arrived Notify =FALSE/ Store Seq. Cnt. in T-PDU Send	Paquet arrivé Rx. T-Header <> Seq. Cnt Notification de données arrivée = FALSE/ Stocker le cmpte séqn dans T-PDU Envoyer
Packet Arrived Rx. T-Header =Seq. Cnt Dup Arrived Notify =FALSE/ Send	Paquet arrivé Rx. T-Header = Seq. Cnt Notification de doublons arrivé = FALSE/ Envoyer

Figure 51 – STD serveur de classe 2

Le service send est invoqué pour les comptes de séquences qui concordent et les comptes de séquences qui diffèrent. Ce comportement est requis pour assurer qu'un nœud ayant envoyé un acquittement qui n'a pas été perdu doit réémettre l'acquittement dans une répétition de tentative.

9.3.7.3.4 Matrice d'événements d'états

Les transitions d'états du transport serveur de classe 2 sont montrées dans le Tableau 276.

Tableau 276 – SEM serveur de classe 2

Evénement	Etat			
	Non-existent (Inexistant)	**Inactif**	**Ready to run (prêt à fonctionner)**	**Running (En marche)**
Create	Passer à **Idle**	Erreur	Erreur	Erreur
Delete	Erreur	Passer à **Non-existent**	Erreur	Erreur
Start	Erreur	Passer à **Ready to Run**	Erreur	Erreur
Stop	Erreur	Aucune action	Passer à **Idle**	Passer à **Idle**
Reset	Erreur	Erreur	Aucune action	Passer à **Ready to Run**
Paquet arrivé En-tête T Rx. <> Seq.Count Données arrivées Notify = TRUE Doublon arrivé Notify = TRUE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU Tx. 2) Envoyer (Implique Ack) 3)Notifier: Données arrivées 4) Passer à **Running**	1) Stocker Seq. Cnt. dans la TPDU Tx. 2) Envoyer (Implique Ack) 3)Notifier: Données arrivées

Evénement	Etat			
	Non-existent (Inexistant)	Inactif	Ready to run (prêt à fonctionner)	Running (En marche)
Paquet arrivé En-tête T-Header = Seq. Count Données arrivées Notify = TRUE Doublon arrivé Notify = TRUE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU Tx. 2) Envoyer (Implique Ack) 3)Notifier: Données arrivées 4) Passer à **Running**	1) Envoyer (Implique Ack) 2) Notifier: Doublon arrivé
Paquet arrivé En-tête T Rx. <> Seq.Count Données arrivées Notify = TRUE Doublon arrivé Notify = FALSE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU Tx. 2) Envoyer (Implique Ack) 3)Notifier: Données arrivées 4) Passer à **Running**	1) Stocker Seq. Cnt. dans la Tx. 2) Envoyer (Implique Ack) 3)Notifier: Données arrivées
Paquet arrivé En-tête T-Header = Seq. Count Données arrivées Notify = TRUE Doublon arrivé Notify = FALSE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU Tx. 2) Envoyer (Implique Ack) 3) arrivées 4) Passer à **Running**	1) Envoyer (implique Ack)
Paquet arrivé En-tête T Rx. <> Seq.Count Données arrivées Notify = FALSE Doublon arrivé Notify = TRUE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU Tx. 2) Envoyer (Implique Ack) 3) Passer à **Running**	1) Stocker Seq. Cnt. dans la TPDU Tx. 2) Envoyer (Implique Ack)
Paquet arrivé En-tête T-Header = Seq. Count Données arrivées Notify = FALSE Doublon arrivé Notify = TRUE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU Tx. 2) Envoyer (Implique Ack) 3) Passer à **Running**	1) Envoyer (Implique Ack) 2) Notifier: Doublon arrivé
Paquet arrivé En-tête T Rx. <> Seq.Count Données arrivées Notify = FALSE Doublon arrivé Notify = FALSE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU Tx. 2) Envoyer (Implique Ack) 3) Passer à **Running**	1) Stocker Seq. Cnt. dans la TPDU Tx. 2) Envoyer (Implique Ack)
Paquet arrivé En-tête T-Header = Seq. Count Données arrivées Notify = FALSE Doublon arrivé Notify = FALSE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU Tx. 2) Envoyer (Implique Ack) 3) Passer à **Running**	1) Envoyer (implique Ack)

9.3.8 Diagrammes d'états de transport – classe 3

9.3.8.1 Fonctions

La classe de transport 3 commence par le comportement de classe 1 et ajoute une connexion de retour qui vérifie que l'application a reçu le paquet. Le compte de séquences qui doit être retourné avec le "verify" (la vérification) est le même que le compte de séquences envoyé avec les données. Cette connexion de retour est utilisée pour identifier le message de vérification.

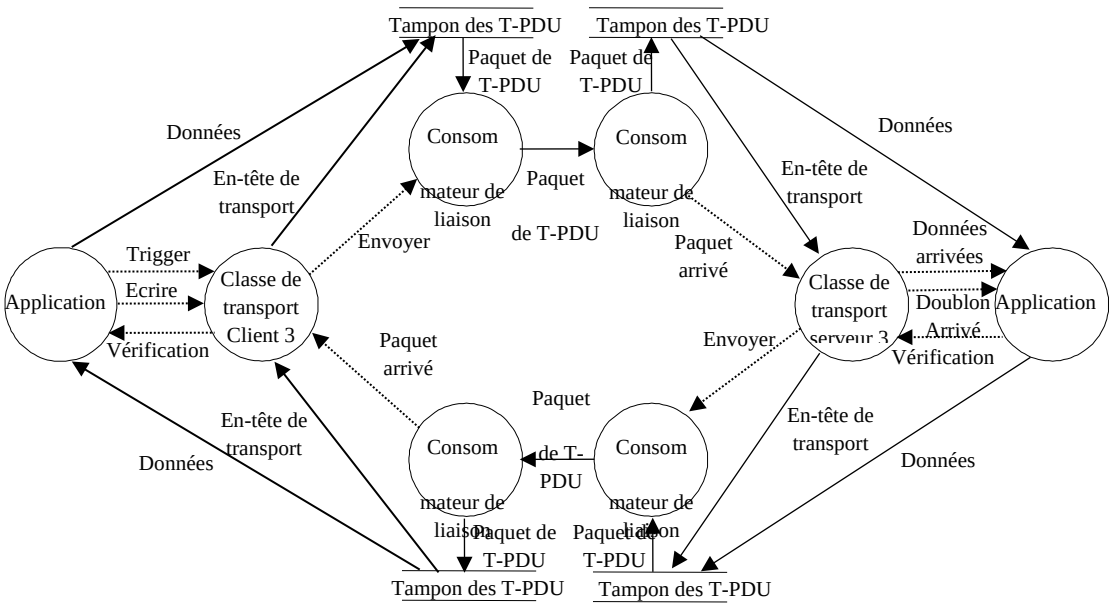
Cette classe de transport permet de notifier à l'application que l'application serveur a répondu au paquet émis. Cette vérification dit à l'application client non seulement que les données sont arrivées, mais aussi que le serveur a lu le tampon et qu'un nouvel échantillon peut être envoyé sans écraser en écriture le tampon.

Le principal avantage de cette classe sur la classe 2 (Acknowledgement) est que la vérification achemine une réponse d'application. L'inconvénient est que le retard pour recevoir la vérification peut être très difficile à prédire.

La classe de transport 3 utilise deux connexions: la connexion dans le sens "émetteur vers cible", qui peut être point à point ou multipoint; et la connexion de retour, qui doit être point à point. Une connexion multipoint en utilisant la classe de transport 3 doit avoir une connexion de retour point à point correspondant à chaque connexion dans le sens "client vers serveur". La vérification à l'application client que l'application serveur a reçu et lu les données émises. Si une application serveur ne vérifier la réception du paquet, l'application client réémet le paquet envoyé le plus récemment; si la connexion client vers serveur est une connexion multipoint, les données sont réémises sur toutes les connexions réseau de la connexion multipoint.

L'application serveur déclenche et envoie la vérification après que le nœud consommateur de la connexion réseau a écrit le paquet dans le tampon des TPDU, après que le transport a lu l'en-tête de transport du paquet et après que l'application a lu le paquet. La vérification envoyée par le serveur signale à l'application client que le tampon de réception des TPDU côté serveur est prêt à accepter le prochain paquet de données.

La Figure 52 montre les actions qui ont lieu au cours d'un transfert de données utilisant la classe de transport client 3 et la classe de transport serveur 3.

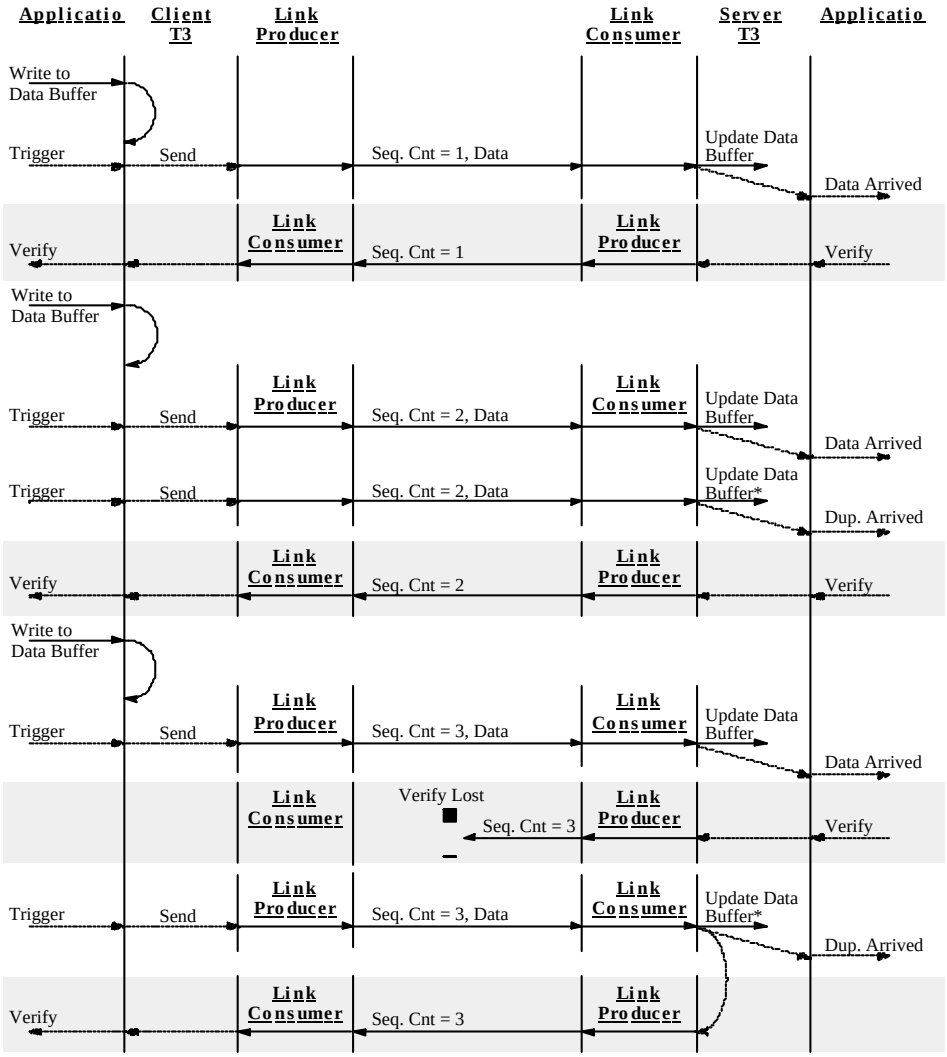


NOTE 1 Les transports client et serveur utilisent chacun deux tampons: un pour la transmission, un autre pour la réception.

NOTE 2 L'application serveur peut retourner des données à l'application client en utilisant la connexion établie pour la vérification.

Figure 52 – Diagramme de flots de données utilisant la classe de transport client 3 et la classe de transport serveur 3

La Figure 53 montre les actions qui ont lieu au cours d'un transfert de données utilisant la classe de transport client 3 et la classe de transport serveur 3. Dans l'exemple décrit, la vérification est la seule donnée transmise du serveur au client. Les zones non ombrées de la Figure 53 indiquent une émission de données dans le sens client vers serveur tandis que les zones ombrées indiquent une émission dans le sens serveur vers client.



* Implementors must decide whether to update the data buffer upon receipt of duplicate data packets, since they can best assess the impact of the additional

Légende

Anglais	Français
Application	Application
Client T3	Client T3
Link Producer	Producteur de liaison
Link Consumer	Consommateur de liaison
Server T3	Serveur T3
Write to Data Buffer	Ecrire dans le tampon de données
Trigger	Déclencher
Send	Envoyer
Seq. Cnt = 1	Compte de séquence = 1
Update Data Buffer	Mettre à jour le tampon de données
Seq. Cnt = 1, Data	Compte de séquence = 1, Données
Data Arrived	Données arrivées
Dup. Arrived	Doublons arrivés
Verify	Vérification
Verify Lost	Vérification perdue
Update Buffer*	Mettre à jour le tampon*

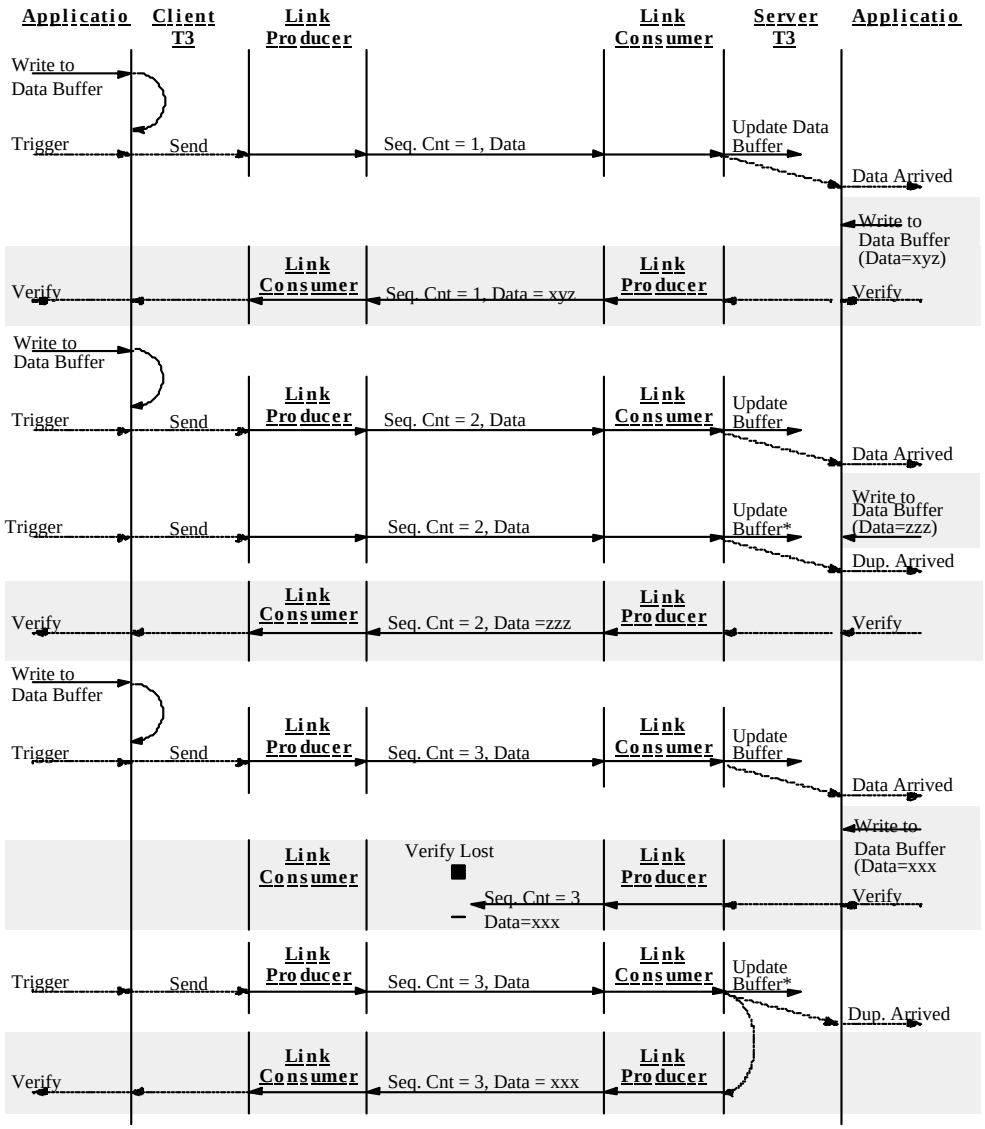
Anglais	Français
* Implementors must decide whether to update the data buffer upon receipt of duplicate data packets since they can best assess the impact of the additional processing.	* Les réalisateurs doivent décider de mettre à jour ou pas le tampon de données à la réception de doublons de paquets de données, car ils peuvent le mieux estimer l'impact du traitement complémentaire.

Figure 53 – Diagramme de séquence de transfert de données utilisant la classe de transport client 3 et la classe de transport serveur 3 sans données retournées

Par moments, l'émission dans le sens serveur vers client comprend la vérification and des données complémentaires. La Figure 54 montre la séquence dans laquelle des actions ont lieu au cours du transfert de données utilisant la classe de transport client 3 et la classe de transport serveur 3 lorsque la vérification et des données sont retournés au client.

NOTE Les données retournées sont écrites de façon asynchrone dans l'accusé de réception des données issu du client.

Les zones non ombrées de la Figure 54 indiquent une émission de données dans le sens client vers serveur tandis que les zones ombrées indiquent une émission dans le sens serveur vers client.



* Implementors must decide whether to update the data buffer upon receipt of duplicate data since they can best assess the impact of the additional processing.

Légende

Anglais	Français
Application	Application
Client T3	Client T3
Link Producer	Producteur de liaison
Link Consumer	Consommateur de liaison
Server T3	Serveur T3
Write to Data Buffer	Ecrire dans le tampon de données
Trigger	Déclencher
Send	Envoyer
Seq. Cnt = 1	Compte de séquence = 1
Update Data Buffer	Mettre à jour le tampon de données
Seq. Cnt = 1, Data	Compte de séquence = 1, Données
Data Arrived	Données arrivées
Dup. Arrived	Doublons arrivés
Write to Data Buffer (Data =xyz)	Ecrire dans le tampon de données (Données=xyz)

Anglais	Français
Write to Data Buffer (Data =zzz)	Ecrire dans le tampon de données (Données=zzz)
Write to Data Buffer (Data =xxx)	Ecrire dans le tampon de données (Données=xxx)
Seg. Cnt = 1, Data=xyz	Seg. Cnt = 1, Data=xyz
Seg. Cnt = 2, Data=zzz	Seg. Cnt = 2, Data=zzz
Seg. Cnt = 3, Data=xxx	Seg. Cnt = 3, Data=xxx
Verify	Vérification
Verify lost Lost	Vérification perdue
Update Buffer*	Mettre à jour le tampon*
* Implementors must decide whether to update the data buffer upon receipt of duplicate data packets since they can best assess the impact of the additional processing.	* Les réalisateurs doivent décider de mettre à jour ou pas le tampon de données à la réception de doublons de paquets de données, car ils peuvent le mieux estimer l'impact du traitement complémentaire.

Figure 54 – Diagramme de séquence de transfert de données utilisant la classe de transport client 3 et la classe de transport serveur 3 sans données retournées

Les utilisations possibles de la classe de transport de 3 comprennent les entrées et sorties de changement d'état, les transferts de blocs [indépendants] asynchrones, les acquittements d'événements, la communication entre dispositifs de commande et appareils d'interface opérateur, les téléchargements vers l'amont, les téléchargements vers l'aval et la messagerie.

9.3.8.2 Classe de transport client 3

9.3.8.2.1 Fonctions

Dans le sens client vers serveur, un transport client de classe 3 a la responsabilité:

- d'accepter le déclencheur de l'application client qu'il a écrit un paquet de données dans le tampon d'émission des TPDU côté client;
- d'écrire un en-tête de transport dans le tampon d'émission des TPDU côté client pour chaque paquet que l'application client y écrit;
- d'initialiser le compte de séquences dans l'en-tête de transport (pour le premier paquet émis) ou d'incrémenter le compte de séquences (pour les paquets ultérieurs de la même émission);
- de déclencher le nœud producteur de la connexion réseau pour produire le paquet TPDU sur la liaison et l'envoyer vers le nœud consommateur de la connexion réseau.

Dans le sens serveur vers client, un transport client de classe 3 a la responsabilité:

- d'accepter la notification issue du nœud consommateur de la connexion réseau de retour qu'il a écrit des données (la vérification de l'instance de transport serveur) dans le tampon de réception des TPDU côté client;
- de localiser et de lire l'en-tête de transport de chaque paquet de données que le nœud consommateur de la connexion réseau de retour écrit dans le tampon de réception des TPDU côté client;
- de notifier à l'application client que le nœud consommateur de la connexion réseau de retour a écrit des données dans le tampon de réception des TPDU côté client.

L'application peut mettre à jour le tampon de données ou redéclencher les données avant que toutes les vérifications n'aient été reçues. Si le transport client est redéclenché alors qu'il attend une vérification, le registre de vérification précédente reçue est effacé. Cela signifie que la classe de transport client doit attendre que tous les consommateurs actifs retournent une vérification avant de faire notification à l'application.

9.3.8.2.2 Etats

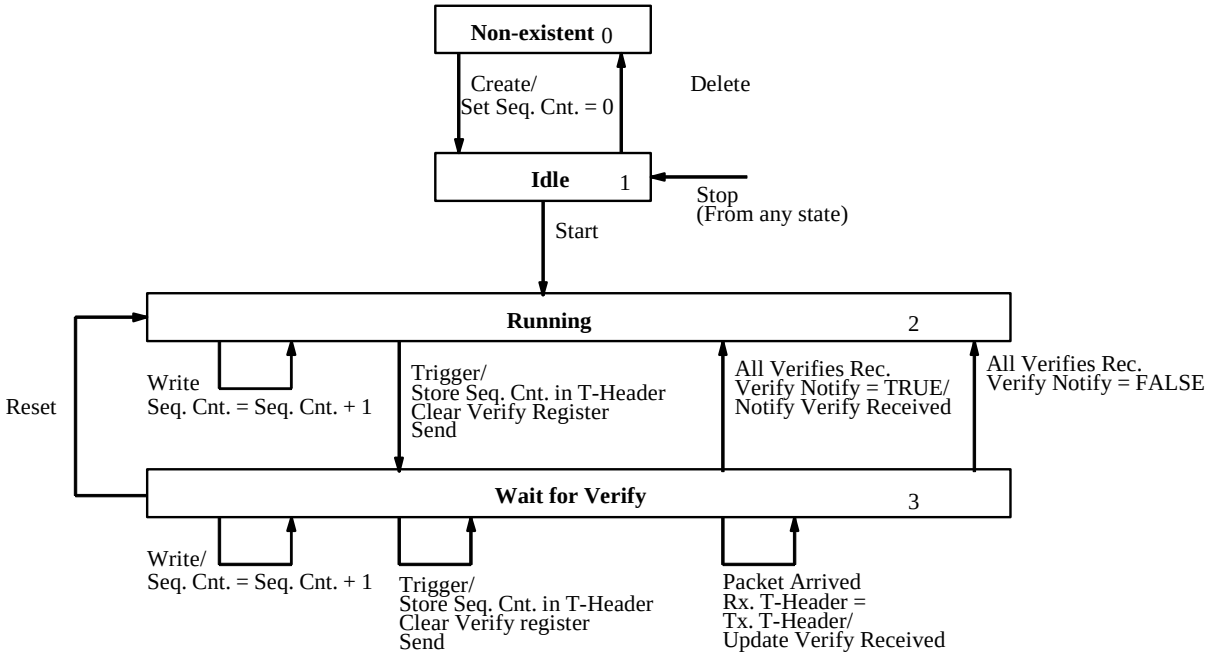
Les états définis et leurs descriptions du client de transport de classe 3 sont énumérés dans le Tableau 277.

Tableau 277 – Etats du client de transport de classe 3

Etat	Description
Non-existent (Inexistant)	L'instance de classe de transport 3 n'existe pas.
Inactif	L'instance de classe de transport 3 a été créée, mais n'a pas encore démarré.
Running (En marche)	L'instance de classe de transport 3 a été créée et a démarré.
Wait to verify	L'instance de la classe de transport 3 a été créée et a démarré, la vérification est en cours.

9.3.8.2.3 Transitions d'états

Les transitions d'états du transport client de classe 3 sont montrées à la Figure 55.



NOTE Le compte de séquences est mis à 0 par le service create. Il est ensuite incrémenté après chaque événement *write* tant qu'il n'a pas atteint une valeur maximale de 2^{16}-1. Après quoi, il repasse par 0.

Légende

Anglais	Français
Non-existent	Inexistant
Delete	Supprimer
Create/ Set Seq. Cnt. = 0	Créer/ Mettre Seq. Cnt. = 0
Idle	Au repos
Stop (From any state)	Arrêter (de n'importe quel état)
Start	démarrer
Running	En marche

Anglais	Français
All Verifies Rec. Verify Notify = FALSE	Toutes les vérifications reçues Notifier vérification = FALSE
All Verifies Rec. Verify Notify = TRUE/ Notify Verify Received	Toutes les vérifications reçues Notifier vérification = TRUE Notifier vérification reçue
Trigger/ Store Seq. Cnt. in T-Header Clear Verify Register Send	Déclencher/ Stocker Seq. Cnt dans T-Header Effacer le registre des vérifications Envoyer
Write	Ecrire
Seq. Cnt. = Seq. Cnt. + 1	Seq. Cnt. = Seq. Cnt. + 1
Reset	Réinitialiser
Wait for Verify	Attendre une vérification
Write/ Seq. Cnt. = Seq. Cnt. + 1	Ecrire Seq. Cnt. = Seq. Cnt. + 1
Trigger/ Store Seq. Cnt. in T-Header Clear Verify register Send	Déclencher/ Stocker Seq. Cnt dans T-Header Effacer le registre des vérifications Envoyer
Packet Arrived Rx. T-Header =Tx. T-Header/ Update Verify Received	Paquet arrivé Rx. T-Header =Tx. T-Header/ Mettre à jour la vérification reçue

Figure 55 – STD client de classe 3

9.3.8.2.4 Matrice d'événements d'états

Les transitions d'états du transport client de classe 3 sont montrées au Tableau 278.

Tableau 278 – SEM client de classe 3

Evénement	Etat			
	Non-existent (Inexistant)	**Inactif**	**Running (En marche)**	**Wait for verify**
Create	1) Seq. Cnt = 0 2) Passer à **Idle**	Erreur	Erreur	Erreur
Delete	Erreur	Passer à **Non-existent**	Erreur	Erreur
Start	Erreur	Passer à **Running**	Erreur	Erreur
Stop	Erreur	Aucune action	Passer à **Idle**	Passer à **Idle**
Reset	Erreur	Erreur	Aucune action	Passer à **Running**
Ecrire	Erreur	Aucune action	Seq. Cnt = Seq. Cnt + 1	Seq. Cnt = Seq. Cnt + 1
Trigger	Erreur	Aucune action	1) Stocker Seq. Cnt. dans la TPDU 2) Effacer le registre des Vérif. 3) Envoyer 4) Passer à **Wait for Verify.**	1) Stocker Seq. Cnt. dans la TPDU 2) Effacer le registre des Vérif. 3) Envoyer

Evénement	Etat			
	Non-existent (Inexistant)	**Inactif**	**Running (En marche)**	**Wait for verify**
Paquet arrivé En-tête T Rx. = En-tête T Rx.	Aucune action	Aucune action	Aucune action	1) Mise à jour le registre des Verify.
Toutes les Vérif. reçues Notify = TRUE	Aucune action	Aucune action	Erreur	1) Notify: Vérif. 2) Passer à **Running**
Toutes les Vérif. reçues Notify = FALSE	Aucune action	Aucune action	Erreur	1) Passer à **Running**

NOTE Le STD et la SEM pour les classes client 2 et 3 sont presque identiques. "Acknowledgment" (acquittement) a remplacé "verification" (vérification).

9.3.8.3 Classe de transport serveur 3

9.3.8.3.1 Fonctions

La classe de transport 3 commence par le comportement de classe 1 et ajoute une connexion de retour qui vérifie que l'application a reçu le paquet. Il est de la responsabilité de l'application de vérifier la réception des données. Le compte de séquences qui doit être retourné avec la vérification est le même que le compte de séquences envoyé avec les données.

L'application est tenue de lire le tampon des TPDU avant d'invoquer les service verify.

Dans le sens client vers serveur, un transport serveur de classe 3 a la responsabilité:

- d'accepter la notification issue du nœud consommateur de la connexion réseau qu'il a écrit un paquet de données dans le tampon de réception des TPDU côté serveur;
- de localiser et de lire l'en-tête de transport de chaque paquet que le nœud consommateur de la connexion réseau écrit dans le tampon de réception des TPDU côté serveur;
- de comparer le compte de séquences de chaque paquet dans le tampon de réception des TPDU côté serveur à celui du paquet précédent;
- de notifier à l'application serveur que le paquet arrivé le plus récemment est un doublon du précédent lorsque leurs comptes de séquences concordent;
- de notifier à l'application serveur que des données ont été écrites dans le tampon de réception des TPDU côté serveur.

Dans le sens serveur vers client, un transport serveur de classe 3 a la responsabilité:

- d'écrire dans le tampon d'émission des TPDU côté serveur l'en-tête de transport des données dans le tampon de réception des TPDU côté serveur;
- de corréler l'en-tête de transport avec n'importe quelle donnée que l'application serveur souhaite envoyer à l'application et a écrite dans le tampon d'émission des TPDU côté serveur;
- de déclencher le nœud producteur de la connexion réseau de retour pour produire le paquet TPDU sur la liaison et l'envoyer vers le nœud consommateur de la connexion réseau de retour.

NOTE Les données issues de l'application peuvent être retournées du serveur vers le client le long de la connexion de vérification.

Le service send est invoqué lorsque verifiy est reçu d'une application ou lorsqu'un doublon d'émission de doublon est reçu dans l'état "running". Le fait de réémettre lorsqu'un paquet vérifié précédemment a été reçu permet de récupérer après une perte de vérifications.

9.3.8.3.2 Etats

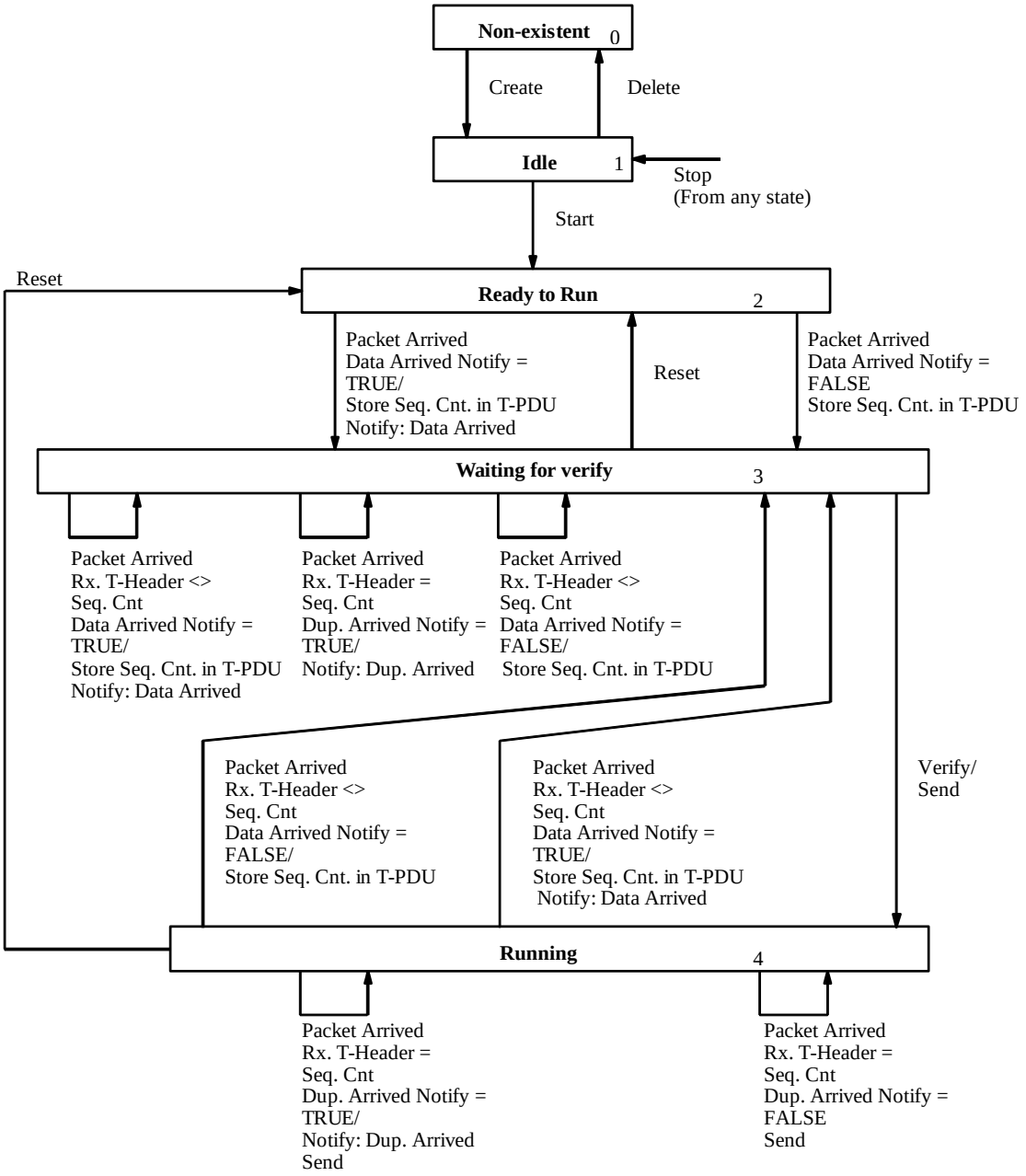
Les états définis et leurs descriptions du serveur de transport de classe 3 sont énumérés dans le Tableau 279.

Tableau 279 – Etats du serveur de transport de classe 3

Etat	Description
Non-existent (Inexistant)	L'instance de classe de transport 3 n'existe pas.
Inactif	L'instance de classe de transport 3 a été créée, mais n'a pas encore démarré.
Read to run	L'instance de la classe de transport 3 a été créée et a démarré, attendant que le premier paquet arrive.
Wait for verify	L'instance de la classe de transport 3 a été créée et a démarré, le premier paquet est arrivé, attendant qu'une vérification soit demandée par l'application.
Running (En marche)	L'instance de la classe de transport 3 a été créée et a démarré, la vérification a été traitée.

9.3.8.3.3 Transitions d'états

Les transitions d'états du transport serveur de classe 3 sont montrées dans la Figure 56.



Légende

Anglais	Français
Non-existent	Inexistant
Create	Créer
Delete	Delete
Idle Start Stop (From any state)	Au Repos Démarrer Arrêter (de n'importe quel état)
Ready to Run	Prêt à fonctionner
Packet Arrived Data Arrived Notify = FALSE Store Seq. Cnt. in T-PDU	Paquet arrivé Notification de données arrivées =FALSE Stocker Seq.Cnt dans T-PDU
Reset	Réinitialiser

Anglais	Français
Packet Arrived Data Arrived Notify =TRUE/ Store Seq. Cnt. in T-PDU Notify: Data Arrived	Paquet arrivé Notification de données arrivées =TRUE Stocker Seq.Cnt dans T-PDU Notifier: Données arrivées
Waiting for verify	Attente de vérification
Packet Arrived Rx. T-Header <> Seq. Cnt Data Arrived Notify = FALSE/ Store Seq. Cnt. in T-PDU	Paquet arrivé Rx. T-Header <> Seq. Cnt Notification de données arrivées = FALSE/ Stocker Seq.Cnt dans T-PDU
Packet Arrived Rx. T-Header = Seq. Cnt Dup. Arrived Notify = TRUE/ Notify: Dup. Arrived	Paquet arrivé Rx. T-Header = Seq. Cnt Notification de doublons arrivée = TRUE/ Notifier: Doublons arrivés
Packet Arrived Rx. T-Header <> Seq. Cnt Data Arrived Notify = TRUE/ Store Seq. Cnt. in T-PDU Notify: Data Arrived	Paquet arrivé Rx. T-Header <> Seq. Cnt Notification de données arrivées =TRUE Stocker Seq.Cnt dans T-PDU Notifier: Données arrivées
Verify/ Send	Vérifier/ Envoyer
Packet Arrived Rx. T-Header <> Seq. Cnt Data Arrived Notify =TRUE/ Store Seq. Cnt. in T-PDU Notify: Data Arrived	Paquet arrivé Rx. T-Header <> Seq. Cnt Notification de données arrivées =TRUE Stocker Seq.Cnt dans T-PDU Notifier: Données arrivées
Packet Arrived Rx. T-Header <> Seq. Cnt Data Arrived Notify = FALSE/ Store Seq. Cnt. in T-PDU	Paquet arrivé Rx. T-Header <> Seq. Cnt Notification de données arrivées =FALSE Stocker Seq.Cnt dans T-PDU
Running	En marche
Packet Arrived Rx. T-Header =Seq. Cnt Dup. Arrived Notify =FALSE Send	Paquet arrivé Rx. T-Header = Seq. Cnt Notification de doublons arrivés =FALSE Envoyer
Packet Arrived Rx. T-Header =Seq. Cnt Dup. Arrived Notify =TRUE/ Notify: Dup. Arrived	Paquet arrivé Rx. T-Header = Seq. Cnt Notification de doublons arrivés =TRUE Notifier: Doublons arrivés
Send	Envoyer
Reset	Réinitialiser

Figure 56 – STD serveur de classe 3

9.3.8.3.4 Matrice d'événements d'états

Les transitions d'états du transport serveur de classe 3 sont montrées dans le Tableau 280.

Tableau 280 – SEM serveur de classe 3

Evénement	Etat				
	Non-existent	Inactif	Ready to run	Waiting for verify	Running (En marche)
Create	Passer à Idle	Erreur	Erreur	Erreur	Erreur
Delete	Erreur	Passer à Non-existent	Erreur	Erreur	Erreur
Start	Erreur	Passer à Ready to Run	Erreur	Erreur	Erreur
Stop	Erreur	Aucune action	Passer à Idle	Passer à Idle	Passer à Idle
Reset	Erreur	Erreur	Aucune action	Passer à Ready to Run	Passer à Ready to Run
Paquet arrivé En-tête T Rx. <> Seq.Count Données arrivées Notify = TRUE Doublon arrivé Notify = TRUE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU 2) Notify: Données arrivées 3) Passer à Waiting for Verify	1) Stocker Seq. Cnt. dans la TPDU 2) Notify: Données arrivées	1) Stocker Seq. Cnt. dans la TPDU 2) Notify: Données arrivées 3) Passer à Waiting for Verify
Paquet arrivé En-tête T Rx. = Seq.Count Données arrivées Notify = TRUE Doublon arrivé Notify = TRUE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU 2) Notify: Données arrivées 3) Passer à Waiting for Verify	1) Notify: Doublon Arrivé	1) Notify: Doublon arrvié 2) Envoyer (Implique Verify)
Paquet arrivé En-tête Rx. <> Seq.Count Données arrivées Notify = TRUE Doublon arrivé Notify = FALSE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU 2) Notify: Données arrivées 3) Passer à Waiting for Verify	1) Stocker Seq. Cnt. dans la TPDU 2) Notify: Données arrivées	1) Stocker Seq. Cnt. dans la TPDU 2) Notify: Données arrivées 3) Passer à Waiting for Verify
Paquet arrivé En-tête T Rx. = Seq.Count Données arrivées Notify = TRUE Doublon arrivé Notify = FALSE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU 2) Notify: Données arrivées 3) Passer à Waiting for Verify	Aucune action	1) Envoyer (implique Verify).
Paquet arrivé En-tête T Rx. <> Seq.Count Données arrivées Notify = FALSE Doublon arrivé Notify = TRUE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU 2) Passer à Waiting for Verify	1) Stocker Seq. Cnt. dans la TPDU	1) Stocker Seq. Cnt. dans la TPDU 2) Passer à Waiting for Verify
Paquet arrivé En-tête T Rx. = Seq.Count Données arrivées Notify = FALSE Doublon arrivé Notify = TRUE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU 2) Passer à Waiting for Verify	1) Notify: Doublon Arrivé	1) Notify: Doublon arrvié 2) Envoyer (Implique Verify)
Paquet arrivé En-tête T Rx. <> Seq.Count Données arrivées Notify = FALSE Doublon arrivé Notify = FALSE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU 2) Passer à Waiting for Verify	1) Stocker Seq. Cnt. dans la TPDU.	1) Stocker Seq. Cnt. dans la TPDU 2) Passer à Waiting for Verify

Evénement	Etat				
	Non-existent	Inactif	Ready to run	Waiting for verify	Running (En marche)
Paquet arrivé En-tête T Rx. = Seq.Count Données arrivées Notify = FALSE Doublon arrivé Notify = FALSE	Aucune action	Aucune action	1) Stocker Seq. Cnt. dans la TPDU 2) Passer à Waiting for Verify	Aucune action	1) Envoyer (implique Verify).
Vérification	Erreur	Aucune action	Aucune action	1) Envoyer (Implique Verify) 2) Passer à Running	Aucune action

9.3.9 Diagrammes d'états de transport – classes 4, 5, 6

NOTE Le contenu du paragraphe de l'édition précédente est devenu obsolète.

9.3.10 Diagrammes d'états de transport – classe 4

NOTE Le contenu du paragraphe de l'édition précédente est devenu obsolète.

9.3.11 Diagrammes d'états de transport – classe 5

NOTE Le contenu du paragraphe de l'édition précédente est devenu obsolète.

9.3.12 Diagrammes d'états de transport – classe 6

NOTE Le contenu du paragraphe de l'édition précédente est devenu obsolète.

10 Diagramme protocolaire de mise en correspondance DLL 1 (DMPM 1)

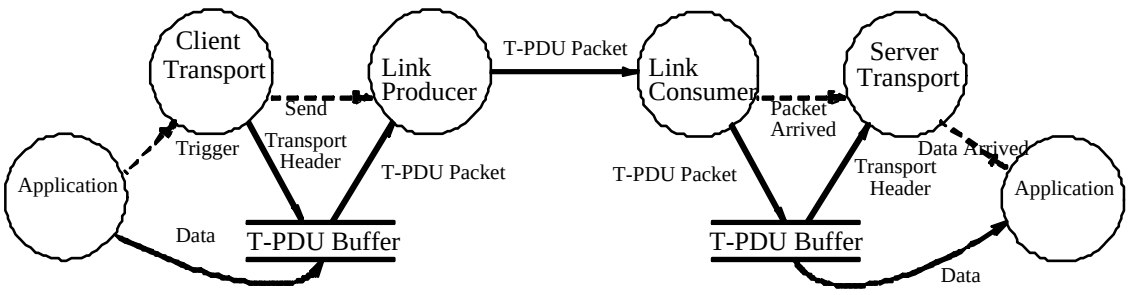
10.1 Généralités

L'Article 10 définit les exigences pour mettre en correspondance la transmission des TPDU dans une couche liaison de données de Type 2.

Le DLL Mapping Protocol Machine (Diagramme protocolaire de mise en correspondance de couche Liaison de données) est commun à tous les types d'AREP.

Les connexions réseau qui s'interfacent à la couche Liaison de données comprennent un ou plusieurs producteurs de liaisons et un ou plusieurs consommateurs de liaisons.

Le flot de données du producteur et du consommateur de liaisons sur une instance de connexion Transport est montré à la Figure 57.



Légende

Anglais	Français
Application	Application

Anglais	Français
Client transport	Transport client
Send	Envoyer
T-PDU Buffer	Tampon des T-PDU
T-PDU Packet	Paquet de T-PDU
Trigger	Déclencher
Data	Données
Transport Header	En-tête de transport
Link Producer	Producteur de liaison
Link Consumer	Consommateur de liaison
Data Arrived	Données arrivées
Server Transport	Transport serveur
Packet arrived	Paquet arrivé

Figure 57 – Diagramme de flot de données pour producteur et un consommateur de données

10.2 Producteur de liaisons

Le producteur de liaisons doit avoir la responsabilité de rechercher des données dans le tampon des TPDU et de les produire sur la liaison de données après que l'événement **send** a été reçu. Les plans de gestion du tampon des TPDU tels que l'écrasement en écriture, la double mise en tampon, la triple mise en tampon, ne sont pas exclus, mais ils ne sont pas spécifiés ici parce qu'ils sont spécifiques à une mise en œuvre.

Le producteur de liaison doit lire le paquet de TPDU dans le tampon des TPDU et l'émettre inchangé. Alors que le tampon des TPDU est montré comme un seul tampon, cela n'implique pas que l'en-tête de transport et les données soient stockés dans des emplacements consécutifs. Une mise en œuvre qui a stocké l'en-tête de transport et les données dans des emplacements non consécutifs est autorisée tant que le producteur de liaisons peut trouver toutes les parties du paquet de TPDU.

L'application, et non le producteur de liaisons, doit avoir la responsabilité d'assurer que le tampon des TPDU a été initialisé avant que le premier déclencheur n'ait été émis.

10.3 Consommateur de liaisons

Le consommateur de liaisons a la responsabilité de recevoir des paquets de TPDU, de les stocker dans le tampon des TPDU et de notifier à la classe de transport serveur qu'un paquet est arrivé. Les plans de gestion du tampon de données tels que l'écrasement en écriture, la double mise en tampon, la triple mise en tampon, ne sont pas exclus, mais ils ne sont pas spécifiés ici parce qu'ils sont spécifiques à une mise en œuvre.

Le consommateur de liaison doit écrire dans le tampon des TPDU le paquet de TPDU inchangé. Alors que le tampon des TPDU est montré comme un seul tampon, cela n'implique pas que l'en-tête de transport et les données soient stockés dans des emplacements consécutifs. Il n'y a pas de restrictions sur les mises en œuvre réelles. Une mise en œuvre qui a stocké l'en-tête de transport et les données dans des emplacement non consécutifs est autorisée tant que le consommateur de liaisons peut trouver les deux parties du paquet de TPDU.

10.4 Définitions de primitive

10.4.1 Primitives échangées entre le diagramme DMPM et le diagramme ARPM

La liste des primitives échangées entre DMPM et ARPM est montrée dans le Tableau 281 et dans le Tableau 282.

Tableau 281 – Primitives émises par l'ARPM vers le diagramme DMPM

Nom de la primitive	Source	Paramètres associés	Fonctions
Send.req	ARPM	Service Dest-id TPDU	Cette primitive est utilisée pour demander au DMPM de transférer la TPDU vers la liaison de données. Elle transmet la TPDU au DMPM comme une DLSDU.

Tableau 282 – Primitives émises par le DMPM vers l'ARPM

Nom de la primitive	Source	Paramètres associés	Fonctions
Packet_Arrived.ind	DMPM	Service Source-id TPDU	Cette primitive est utilisée pour transmettre la TPDU reçue comme unité de données de services de couche Liaison de données vers l'ARPM désigné.

10.4.2 Paramètres de primitives d'ARPM/DMPM

Les paramètres utilisés avec les primitives échangées entre l'ARPM et le DMPM sont décrits dans le Tableau 283.

Tableau 283 – Paramètres utilisés avec les primitives échangées entre l'ARPM et le DMPM

Nom de paramètre	Description
TPDU	Ce paramètre transporte la PDU de transport contenant les données utilisateur et l'en-tête de Transport.
DL-SDU	Ce paramètre transporte la SDU de liaison de données reçue.

10.4.3 Primitives échangées entre la couche Liaison de données et le diagramme DMPM

Les services de demande ("request") et de confirmation ("confirm") utilisés pour place en file d'attente les Lpackets avec la couche Liaison de données doivent avoir la forme:

NOTE Les primitives montrées dans le Tableau 284 et leurs paramètres sont définis dans le détail dans les articles de la CEI 61158-3-2.

Tableau 284 – Primitives échangées entre la couche Liaison de données et le DMPM

Nom de la primitive	Source	Paramètres associés
DL_generic_Tag.req	ARPM	Request_DLS_user_id (handle) user_data [] QoS (priority) generic_TAG
DL_generic_Tag.cnf	Couche Liaison de données	Request_DLS_user_id (handle) TX_status
DL_generic_Tag.ind	Couche Liaison de données	Request_DLS_user_data [] generic_TAG

Nom de la primitive	Source	Paramètres associés
DL_fixed_Tag.req	ARPM	Request_DLS_user_id (handle) user_data [] QoS (priority) fixed_TAG destination-DLE-ID
DL_fixed_Tag.cnf	Couche Liaison de données	Request_DLS_user_id (handle) TX_status
DL_fixed_Tag.ind	Couche Liaison de données	user_data [] fixed_TAG source-DLE-ID
DL_tone.ind	Couche Liaison de données	DLS_cycle

10.4.4 Paramètres des primitives de DMPM/Couche Liaison de données

Les paramètres utilisés avec les primitives échangées entre le DMPM et la couche Liaison de données sont identiques à ceux qui sont définis dans la Définition de services de couche Liaison de données (CEI 61158-3-2).

Les paramètres sont décrits dans le Tableau 285.

Tableau 285 – Paramètres utilisés avec les primitives échangées entre DMPM et la couche Liaison de données

Nom de paramètre	Description
Request_DLS_user_id	Fournit un moyen local d'apparier la primitive "confirm" résultante avec la primitive "request" causale.
User_data	Données à émettre entre utilisateurs de DLS ("Data-Link Service" service de liaison de données) sans altération par le fournisseur de DLS.
QoS	Sélectionne l'une des priorités suivantes: urgent scheduled high low
Generic_TAG	Identification de connexion ou adresse de point de service identifiant le(s) DLSAP ("data link service access point" point d'accès aux services de liaison de données) distant(s) au(x)quel(s) le DLS est à fournir.
Fixed_TAG	Spécifie le point de service de destination dans le poste identifié par l'adresse DLS-Destination-DLE-ID.
Destination-DLE-ID	Adresse identifiant le poste de destination pour le message utilisant son adresse d'ID de MAC
Source-DLE-ID	Adresse identifiant le poste local à partir duquel la DLSDU à étiquette fixe a été envoyée. Il s'agit de l'adresse d'ID de MAC du poste sur une liaison locale.
TX_status	Statut de l'émission demandée: a) "OK" le message a été envoyé avec succès; b) "TxAbort" le processus d'envoi a échoué; c) "Flushed" le message a été retiré de la file d'attente en cours avant d'être envoyé.
DLS_cycle	Compte d'intervalles pour le Temps de mise à jour de réseau (NUT) qui a juste été reçu dans les limites du cycle global des intervalles d'accès programmés.

10.4.5 Network connection ID (ID de connexion réseau)

L'ID de connexion réseau (Network Connection ID) doit être une valeur de 32 bits construite comme suit:

- les bits 31 à 24 doivent être réservés et doivent être mis à zéro;

- les bits 23 à 16 doivent contenir un node MAC ID dépendant du type de connexion sélectionné;
- les bits 15 à 0 doivent contenir le numéro de connexion de 16 bits.

Le Tableau 286 montre des options pour la sélection de l'ID de connexion.

Tableau 286 – Sélection de l'ID de connexion

Type de connexion	ID MAC	Connection number (Numéro de connexion)
Multipoint	MAC du producteur	Unique par appareil
Point à Point	MAC du consommateur	Unique par appareil

L'étiquette générique pour les Lpackets échangés sur une connexion et décrits dans la CEI 61158-4-2 doit être construite comme suit, en fonction du contenu des champs CID:

- le premier octet émis de l'étiquette générique doit contenir le MAC ID;
- le deuxième octet émis de l'étiquette générique doit contenir l'octet de poids le plus faible du numéro de connexion;
- le troisième octet émis de l'étiquette générique doit contenir l'octet de poids le plus fort du numéro de connexion.

Pour garantir que les ID de connexion sont uniques sur une liaison, les règles suivantes doivent être utilisées pour choisir les CID:

- le producteur doit choisir le CID pour un transport multipoint;
- le consommateur doit choisir le CID pour un transport point à point;
- sur une liaison CP 2/1, le nœud chargé du choix du CID (soit le producteur, soit le consommateur) doit insérer son MAC ID dans les bits 23 à 16 du CID du Forward_Open;
- un CID multipoint ne doit pas être réutilisé tant que les connexions associées au CID n'ont pas été fermées ou temporisées;
- le fait de recevoir un paquet à étiquette fixe "I'm alive" issu d'un nœud doit fermer immédiatement toutes les connexions avec l'appareil en question.

10.5 Diagramme d'états DMPM

10.5.1 Etats du diagramme DMPM

10.5.1.1 Producteur de liaison

10.5.1.1.1 Etats du producteur de liaisons

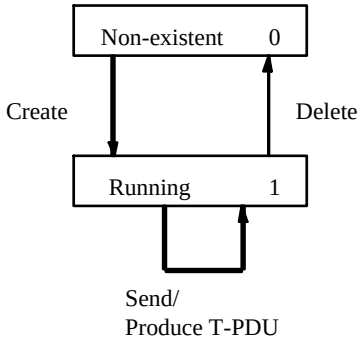
Les états définis et leurs descriptions du Producteur de liaisons sont énumérés dans le Tableau 287 ci-dessous:

Tableau 287 – Etats du producteur de liaisons

Etat	Description
Non-existent (Inexistant)	Le Producteur de liaisons n'existe pas.
Running (En marche)	Le Producteur de liaison a été créé et fonctionne.

10.5.1.1.2 Diagramme de transitions d'états

La Figure 58 montre le diagramme de transitions d'états d'un producteur de liaisons.



Légende

Anglais	Français
Non-existent	Inexistant
Create	Créer
Delete	Supprimer
Running	En marche
Send/ Produce T-PDU	Envoyer/ Produire une T-PDU

Figure 58 – Diagramme de transitions d'états pour un producteur de liaisons

10.5.1.1.3 Matrice d'événements d'états

La matrice d'événements d'états du Producteur de liaison est montrée dans le Tableau 288.

Tableau 288 – Matrice d'événements d'états du producteur de liaisons

Evénement	Etat	
	Non-existent (Inexistant)	Running (En marche)
Create	Passer à **Running**	Erreur
Delete	Erreur	Passer à **Non-existent**
Envoyer	Erreur	1) Produire une TPDU

10.5.1.2 Consommateur de liaisons

10.5.1.2.1 Etats du consommateur de liaisons

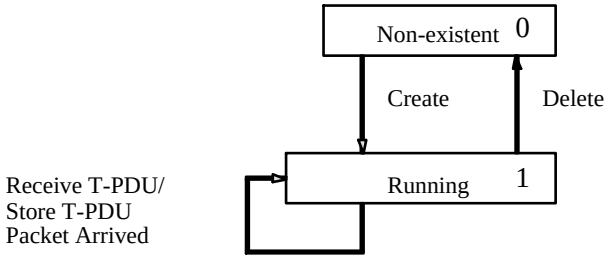
Les états définis et leurs descriptions du Consommateur de liaisons sont énumérés dans le Tableau 289 ci-dessous:

Tableau 289 – Etats du consommateur de liaisons

Etat	Description
Non-existent (Inexistant)	Le Producteur de liaisons n'existe pas.
Running (En marche)	Le Producteur de liaison a été créé et fonctionne.

10.5.1.2.2 Diagramme de transitions d'états

La Figure 59 montre le diagramme de transitions d'états d'un consommateur de liaisons.



Légende

Anglais	Français
Non-existent	Inexistant
Create	Créer
Delete	Supprimer
Running	En marche
Receive T-PDU	Recevoir une T-PDU
Store T-PDU	Stocker la T-PDU
Packet Arrived	Packet Arrived (Paquet arrivé)

Figure 59 – Diagramme de transitions d'états pour un consommateur de liaisons

10.5.1.2.3 Matrice d'événements d'états

La matrice d'événements d'états du Producteur de liaison est montrée dans le Tableau 290.

Tableau 290 – Matrice d'événements d'états du consommateur de liaisons

Evénement	Etat	
	Non-existent (Inexistant)	Running (En marche)
Create	Passer à **Running**	Erreur
Delete	Erreur	Passer à **Non-existent**
Receive TPDU (Recevoir une TPDU)	Erreur	Stocker la TPDU Packet Arrived (paquet arrivé)

10.5.2 Fonctions utilisées par le diagramme DMPM

Ce bus de terrain n'a pas de fonctions spécifiées.

10.6 Sélection des services de couche Liaison de données

Ce bus de terrain prend en charge uniquement le service de couche liaison de données dont les primitives sont indiquées en 10.4.3.

11 Diagramme protocolaire de mise en correspondance DLL 2 (DMPM 2)

11.1 Généralités

L'Article 11 définit les exigences pour la mise en correspondance de l'émission des TPDU sur un réseau Ethernet-TCP/UDP/IP, au lieu de la couche Liaison de données de Type 2, en utilisant la suite de protocoles TCP/UDP/IP et le protocole d'encapsulation défini en 4.3.1.

11.2 Mise en correspondance des PDU de l'UCMM

11.2.1 Généralités

Les PDU de l'UCMM doivent être émises sur une connexion TCP/IP, en utilisant le protocole d'encapsulation défini en 11.8 et 4.3.1 (commande SendRRData).

Une demande UCMM doit être formatée comme cela est montré dans le Tableau 291.

Tableau 291 – Demande UCMM

Structure	Nom de champ		Type de données	Valeur du champ
En-tête d'encapsulation	Command		UINT	SendRRData (0x6F)
	Length		UINT	Longueur d'une portion de données spécifique à la commande
	Session handle		UDINT	Identification retournée par RegisterSession
	Status		UDINT	0
	Sender Context		ARRAY of 8 USHORT	Choisi par l'expéditeur
	Options		UDINT	0
Données spécifiques à la commande	Interface handle		UDINT	0
	Timeout		UINT	Toute valeur (ignoré par la cible)
	Encapsulated packet (dans le format de paquet normalisé)	Item count	UINT	2 (indique 1 élément adresse, 1 élément données)
		Address Type ID	UINT	0 (élément adresse nul)
		Address length	UINT	0
		Data Type ID	UINT	0x00B2 (Unconnected Data Item)
		Data length	UINT	Longueur du champ suivant en octets (longueur du paquet de demande MR)
		MR request packet	ARRAY of USINT	Champ contenant un paquet de demande Message Router de Type 2

Une réponse UCMM doit être formatée comme cela est montré dans le Tableau 292.

Tableau 292 – Réponse UCMM

Structure	Nom de champ	Type de données	Valeur du champ
En-tête d'encapsulation	Command	UINT	SendRRData (0x6F)
	Length	UINT	Longueur d'une portion de données spécifique à la commande
	Session handle	UDINT	Identification retournée par RegisterSession
	Status	UDINT	0
	Sender Context	ARRAY of 8	Copié de la demande UCMM

Structure	Nom de champ		Type de données	Valeur du champ
			USHORT	correspondante
	Options		UDINT	0
Données spécifiques à la commande	Interface handle		UDINT	0
	Timeout		UINT	Toute valeur (ignoré par le destinataire)
	Encapsulated packet (dans le format de paquet normalisé)	Item count	UINT	2 (indique 1 élément adresse, 1 élément données)
		Address Type ID	UINT	0 (élément adresse nul)
		Address length	UINT	0
		Data Type ID	UINT	0x00B2 (Unconnected Data Item)
		Data length	UINT	Longueur du champ suivant en octets (longueur du paquet de réponse MR)
		Paquet de réponse MR	ARRAY of USINT	Champ contenant un paquet de réponse Message Router de Type 2

NOTE Le Paragraphe 4.3.2 définit la format valide pour les commandes d'encapsulation SendRRData et autres.

La taille maximale des éléments du paquet de demande MR ou de réponse MR dans la commande SendRRData est de 504 octets (la taille maximale correspond à la couche Liaison de données de Type 2), afin de garantir que le message UCMM peut traverser toutes les liaisons dans un chemin de réseau de Type 2. Les appareils peuvent prendre en charge le service Large_Forward_Open afin de permettre un accès plus efficace à d'importantes quantités de données d'application.

Lorsqu'une cible reçoit un valeur SendRRData supérieure à la taille maximale, elle doit retourner une réponse d'erreur avec le code d'état d'encapsulation 0x65 (la cible a reçu un message de longueur non valide). Le code d'état 0x03 est admis pour les mises en œuvre existantes, mais doit être considéré comme "déconseillé". Les nouvelles mises en œuvre doivent utiliser 0x65.

11.2.2 Exigences communes pour les PDU du gestionnaire de Connection Manager

11.2.2.1 Généralités

Le Paragraphe 11.2.2 contient les exigences spécifiques pour les paramètres de Connection Manager en cas d'encapsulation TCP/IP (les paramètres de Connection Manager sont décrits dans leur ensemble en 4.1.6).

11.2.2.2 Type de connexion

Le type de connexion doit être NULL, MULTICAST ou POINT2POINT. Le type de connexion MULTICAST ne doit être pris en charge que pour les connexions de classe de transport 0 et de classe de transport 1.

11.2.2.3 Priorité

La priorité doit être LOW (faible), HIGH (élevée), SCHEDULED (planifiée) ou URGENT (urgente), URGENT constituant la priorité la plus forte. Le Paragraphe 11.7 définit le comportement QoS (Quality of Service) relatif aux différents niveaux de priorité de Type 2.

11.2.2.4 Type de déclencheur

Le type de déclencheur doit être CYCLIC, CHANGE_OF_STATE ou APPLICATION. Les connexions des classes de transport 0 et 1 qui utilisent le déclenchement CHANGE_OF_STATE doivent utiliser le segment temps de neutralisation de production segment (voir 4.1.9.5).

11.2.2.5 Taille de connexion

La taille de la connexion ne doit pas être supérieure à 65 511 octets.

NOTE La demande Forward_Open limite la taille de connexion à 511 octets; cependant, la demande facultative Large_Forward_Open autorise des tailles de connexion plus grandes.

11.2.2.6 Temporisation de demande de connexion

Pour établir de manière fiable une connexion qui s'étende à une liaison TCP/IP, la temporisation de demande de connexion doit être suffisamment grande pour permettre d'établir la connexion, ce qui pourrait impliquer de résoudre un nom d'hôte ou de passer par plusieurs passerelles.

En raison de la grande variation des traitements de demande de connexion sur TCP/IP, les routeurs de Type 2 dans le chemin de connexion ne doivent rien soustraire de la temporisation de demande de connexion.

11.2.2.7 Chemin de connexion

La partie adresse de liaison d'un segment de chemin de connexion TCP/IP doit être codée au sein d'un segment de port comme spécifié en 4.1.9.3.

11.2.2.8 ID de connexion réseau

L'ID de connexion réseau doit être un identificateur de 32 bits ayant un sens pour l'appareil qui le choisit. L'ID de connexion réseau peut ne pas être subdivisé en champs spécifiques.

En général, l'appareil consommateur sélectionne l'ID de connexion réseau pour une connexion point à point tandis que l'appareil producteur sélectionne l'ID de connexion réseau pour une connexion multipoint. Le Tableau 293 montre quel appareil, Target (cible) ou Originator (émetteur) doit choisir les ID de connexion réseau dans le sens T⇒O et dans le sens O⇒T.

Tableau 293 – Sélection de l'ID de connexion réseau

Type de connexion	Quel ID de connexion réseau	Qui choisit l'ID de connexion
Point-à-point	Emetteur -> Cible	Cible
	Cible -> Emetteur	Emetteur
Multidiffusion	Emetteur -> Cible	Emetteur
	Cible -> Emetteur	Cible

L'ID de connexion réseau ne doit pas être réutilisé jusqu'à ce que la connexion ait été fermée ou ait temporisé. Lorsqu'un appareil redémarre, il ne doit pas réutiliser les ID de connexion réseau issus de connexions ouvertes précédemment tant que ces connexions n'ont pas été fermées ou n'ont pas été temporisées. Un ID de connexion spécifique ne doit pas être réutilisé tant qu'il existe la possibilité que des paquets avec cet ID de connexion soient présents dans le réseau.

11.2.3 PDU Forward_open pour les connexions de classe 2 et classe 3

La PDU Forward_open pour les connexions de classe 2 et de classe 3 doit être envoyée sur une connexion TCP en utilisant la commande SendRRData définie en 4.3.1.

Les éléments d'informations Sockaddr inclus dans une demande Forward_open ou une PDU Forward_open_response PDU qui établit une connexion de classe 2 ou de classe 3 doivent être ignorés. Lorsque des éléments d'informations Sockaddr sont inclus, il est recommandé que l'émetteur définisse les champs sin_port et sin_addr sur 0.

11.2.4 Forward_open pour les connexions de classe 0 et de classe 1

11.2.4.1 Utilisation de TCP

La PDU Forward_Open pour les connexions de classe de transport 0 et 1 doit être envoyée via une connexion TCP utilisant la commande SendRRData définie en 4.3.1.

11.2.4.2 Utilisation générale des éléments d'informations Sockaddr

En tant que partie intégrante du dialogue de Forward_Open, le producteur et le consommateur doivent échanger les numéros de port UDP et l'adresse IP de multidiffusion (pour les connexions multipoints) nécessaires pour envoyer des données connectées des classes de transport 0 et 1. L'élément Sockaddr Info défini en 4.3.1 doit être utilisé pour coder les numéros de port UDP et l'adresse IP de multidiffusion. L'utilisation de l'élément Sockaddr Info varie selon que la connexion est multipoint ou point à point et selon que l'émetteur de la connexion ou la cible de la connexion est le producteur.

11.2.4.3 Placement et erreurs de l'élément d'information Sockaddr

Les éléments d'informations Sockaddr doivent être placés après les données de la demande Forward_Open et/ou de la réponse Forward_Open dans la commande/réponse SendRRData. Doivent être considérés comme des erreurs les cas où un élément d'information Sockaddr n'est pas présent pour une connexion multipoint, n'a pas la valeur correcte sin_family de AF_INET = 2 ou spécifie des valeurs de champ qui sont illégales pour l'utilisation voulue.

Le destinataire d'une demande Forward_open_request doit retourner une réponse Forward_open_response avec un code d'état 0x01 et un état étendu 0x205 (erreur de paramètre dans un service non connecté) pour les éléments d'informations Sockaddr contenant des erreurs spécifiées dans le paragraphe précédent. Le destinataire d'une réponse Forward_open contenant des éléments d'informations Sockaddr avec des erreurs doit considérer que la réponse entière est erronée.

11.2.4.4 Utilisation de l'élément d'information Sockaddr pour les connexions multipoints

Dans le cas des connexions multipoints, le producteur doit choisir une adresse de multidiffusion IP à laquelle envoyer les données connectées. Le numéro de port doit être le numéro de port UDP enregistré (0x08AE) attribué par l'IANA. Le consommateur multipoint doit recevoir les données connectées sur le numéro de port enregistré. Un élément Sockaddr Info doit être envoyé avec la demande Forward_Open (si le type de la connexion O⇒T est multipoint) ou avec la réponse Forward_Open (si le type de la connexion T⇒O est multipoint). L'adresse IP de multidiffusion choisie doit être codée à l'aide du champ sin_addr de l'élément Sockaddr Info. Le champ sin_port de l'élément Sockaddr Info doit être défini sur 0x08AE et traité par le destinataire comme "sans influence".

11.2.4.5 Utilisation de l'élément Sockaddr Info pour les connexion point à point

Pour les connexions point à point, le consommateur point à point doit choisir un numéro de port UDP auquel les données connectées doivent être envoyées. Le producteur point à point doit envoyer les données connectées au numéro de port choisi par le consommateur point à

point. Il est recommandé que le consommateur utilise le numéro de port enregistré (0x08AE); toutefois, le consommateur peut également choisir un autre numéro de port.

NOTE Utiliser un numéro de port autre que 0x08AE a des répercussions potentielles sur la configuration et la gestion QoS (Quality of Service) dans les appareils de l'infrastructure. Les commutateurs qui ont été configurés pour donner la priorité aux paquets avec le numéro de port 0x8AE risquent de ne pas prioriser les paquets de Type 2 avec un numéro de port différent.

Si le consommateur point à point choisit un numéro de port autre que 0x08AE, alors le consommateur point à point doit envoyer un élément Sockaddr Info indiquant le numéro de port choisi. L'élément Sockaddr Info doit être envoyé avec la demande Forward_open (si le type de la connexion T⇒O est point à point) ou avec la réponse Forward_open (si le type de la connexion O⇒T est point à point). Le champ sin_port de l'élément Sockaddr Info doit contenir le numéro de port sélectionné par le consommateur.

Si le consommateur choisit d'utiliser le port enregistré 0x08AE pour la réception des données connectées, le consommateur n'est pas dans l'obligation d'envoyer un élément Sockaddr Info. Le consommateur peut par ailleurs inclure un élément Sockaddr Info comme cela est décrit dans le paragraphe précédent, avec le champ sin_port contenant le numéro de port enregistré.

Pour un connexion point à point, le champ sin_addr de l'élément Sockaddr Info doit être traité comme "sans influence" par le destinataire de la demande Forward_open ou de la réponse Forward_open. Il est recommandé que l'émetteur définisse le champ sin_addr sur zéro.

11.2.4.6 Récapitulatif de l'utilisation de l'élément Sockaddr Info pour les connexions de classe 0 et 1

Le Tableau 294 montre l'utilisation des éléments Sockaddr Info dans la demande Forward_Open et la réponse Forward_Open de la classe de transport 0et 1. Dans les cas où l'élément est "ignoré si présent", les appareils ne doivent pas vérifier le contenu de l'élément Sockaddr Info pour s'assurer de sa validité.

Tableau 294 – Utilisation de l'élément Sockaddr Info

Type de connexion	Service	Eléments Sockaddr Info
Point-Point O⇒T Multipoint T⇒O	Forward_Open request	O⇒T ignoré si présent T⇒O ignoré si présent
	Forward_Open response	Elément O⇒T facultatif (numéro de port enregistré utilisé si élément non présent) Elément T⇒O obligatoire
Multipoint O⇒T Point-Point T⇒O	Forward_Open request	Elément O⇒T obligatoire Elément T⇒O facultatif (numéro de port enregistré utilisé si élément non présent)
	Forward_Open response	O⇒T ignoré si présent T⇒O ignoré si présent
Point-Point O⇒T Point-Point T⇒O	Forward_Open request	Elément O⇒T ignoré si présent Elément T⇒O facultatif (numéro de port enregistré utilisé si élément non présent)
	Forward_Open response	Elément O⇒T facultatif (numéro de port enregistré utilisé si élément non présent) Elément T⇒O ignoré si présent
Multipoint O⇒T Multipoint T⇒O	Forward_Open request	Elément O⇒T obligatoire Elément T⇒O ignoré si présent
	Forward_Open response	O⇒T ignoré si présent Elément T⇒O obligatoire

11.2.4.7 Mise en correspondance de connexions à des adresses IP de multidiffusion

Bien que cela ne soit pas exigé, il est recommandé que les producteurs utilisent une adresse IP unique de multidiffusion pour chaque connexion multipoint active.

NOTE En fonction de la mise en œuvre, cela peut réduire la quantité de filtrage de connexion de la part du consommateur. Cela permet aussi au consommateur de gérer plus uniformément les données connectées entrantes provenant de plusieurs connexions.

Comme une adresse IP unique de multidiffusion par connexion multipoint n'est pas exigée, les consommateurs doivent pouvoir gérer la situation dans laquelle des paquets provenant de plusieurs connexions multipoints sont envoyés vers la même adresse IP de multidiffusion. Les consommateurs doivent pourvoir filtrer les paquets entrants selon l'ID de connexion et l'adresse IP source (voir 11.3.4 pour les exigences relatives au filtrage).

11.2.4.8 Réalisation d'une connexion multipoint (informative)

Après réception de la réponse Forward_Open, il convient que les appareils Ethernet consommateurs rejoignent le groupe IP de multidiffusion afin de recevoir les datagrammes IP de multidiffusion. La méthode exacte pour ce faire dépend de l'interface de programmation d'application TCP/IP qui est utilisée sur l'appareil.

11.2.4.9 Limitation de portée de multidiffusion IP et attribution d'adresses IP de multidiffusion

11.2.4.9.1 Généralités

Les Paragraphes 11.2.4.9.2 à 11.2.4.9.5 traitent deux questions qui doivent être considérées lors de la mise en œuvre de connexions multipoints sur Ethernet:

- La limitation de portée de la multidiffusion IP se rapporte à la pratique consistant à limiter l'étendue de la propagation à travers le réseau d'un datagramme de multidiffusion donné;
- L'attribution d'adresses IP de multidiffusion se rapporte au problème de savoir comment les applications sélectionnent les adresses IP de multidiffusion qui sont utilisées pour envoyer et recevoir des datagrammes de multidiffusion IP.

NOTE Une fois que l'architecture d'attribution en multidiffusion IP devient une norme et est déployée, les exigences détaillées dans les Paragraphes 11.2.4.9.2 à 11.2.4.9.5 peuvent être mises à jour selon le cas.

11.2.4.9.2 Pratiques de limitation de portée de multidiffusion IP (informative)

En général, la plupart des réseaux déployés actuellement utilisent le "TTL scoping" conjointement à la configuration du routeur et/ou du commutateur pour confiner le trafic de multidiffusion dans les frontières de réseau souhaitées.

"TTL scoping" se rapporte à la pratique consistant à utiliser le champ "Time to Live" (TTL, durée de vie) dans l'en-tête IP pour limiter le nombre de sauts réseau sur lesquels le paquet de multidiffusion est propagé. Lorsqu'il envoie un datagramme de multidiffusion IP, un hôte peut mettre le champ TTL de l'en-tête IP à une valeur appropriée en fonction de l'étendue à laquelle il convient que le diagramme se propage. A mesure que le datagramme est acheminé à travers le réseau, chaque saut décrémente le champ TTL. Les routeurs peuvent être configurés avec des seuils de TTL afin qu'ils ne transmettent pas un paquet si la valeur de TTL n'est pas supérieure au seuil.

Un datagramme de multidiffusion avec une valeur de TTL initiale de 1 limite le datagramme au sous-réseau local. D'autres valeurs de TTL communes sont 16 pour la multidiffusion au sein d'un site et 64 pour la multidiffusion au sein d'une région.

En plus du "TTL scoping", les protocoles d'acheminement multidiffusion et autres méthodes sont communément utilisés pour réguler la propagation du trafic multidiffusion. Les routeurs prennent communément en charge les protocoles de multidiffusion tels que PIM, DVMRP. Les

commutateurs qui mettent en œuvre "IGMP snooping" (écoute IGMP) peuvent limiter les paquets de multidiffusion envoyés sur un port vers les seules adresses de multidiffusion pour lesquelles l'appareil final a émis un message d'adhésion comme membre de l'IGMP ("Internet group management protocol", protocole de gestion de groupe Internet). La configuration des commutateurs et des routeurs est habituellement effectuée par un personnel bien informé.

11.2.4.9.3 Pratiques d'allocation d'adresses de multidiffusion IP (informative)

L'espace entier d'adresses de multidiffusion IP se situe dans la plage de 224.0.0.0 à 239.255.255.255. L'IANA, autorité chargée de l'assignation des numéros Internet (www.iana.org), a la responsabilité de l'attribution de l'espace d'adresses de multidiffusion IP. Des adresses de multidiffusion IP ont été attribuées à des organisations particulières et pour des protocoles particuliers. En outre, il y a un large bloc d'adresses de multidiffusion IP allouées pour la multidiffusion "de portée limitée administrativement", à partir duquel une suite de protocoles d'allocation et de limitation de portée est développée par l'Internet Engineering Task Force (IETF, Groupe de travail d'ingénierie Internet). La plage des adresses de portée limitée administrativement va de 239.0.0.0 à 239.255.255.255 (et elle est en outre divisée en plages supplémentaires).

Actuellement, il n'y a malencontreusement pas de mécanismes normalisés largement répandus pour allouer et affecter des adresses multidiffusion à des applications. Par exemple, lorsqu'un administrateur réseau déploie une application vidéo en streaming, l'application aura son propre mécanisme spécifique pour affecter les adresses de multidiffusion IP.

11.2.4.9.4 Limitation de portée de multidiffusion pour Type 2

Par défaut, les appareils de Type 2 doivent utiliser une valeur de TTL égale à 1 pour les paquets de multidiffusion des classes de transport 0 et 1. L'utilisation d'une valeur de TTL de 1 empêche que les paquets de multidiffusion se propagent au-delà du sous-réseau local. Lorsque la valeur de TTL est égale à 1, le producteur et le consommateur de Type 2 doivent être tous les deux sur le même sous-réseau.

Les appareils de Type 2 sont fortement invités à prendre en charge la configuration explicite de la valeur de TTL pour les paquets de multidiffusion IP. S'ils la prennent en charge, les appareils doivent utiliser l'objet Interface TCP/IP (classe 0xF5) comme mécanisme pour configurer la valeur de TTL. Lorsqu'une valeur de TTL supérieure à 1 est configurée, le producteur et le consommateur peuvent être sur des sous-réseaux différents.

Si la valeur de TTL n'a pas été configurée pour être supérieure à 1 et si une demande de connexion multipoint est reçue d'un émetteur sur un sous-réseau différent, l'appareil doit retourner le statut général 0x01 et le statut étendu 0x813 dans la réponse Forward_Open.

Lorsque la valeur de TTL est configurée explicitement, elle doit être utilisée pour tous les paquets de multidiffusion de Type 2.

11.2.4.9.5 Allocation d'adresse de multidiffusion pour Type 2

11.2.4.9.5.1 Généralités

Deux mécanismes sont définis afin d'attribuer des adresses IP multidiffusion pour des paquets multidiffusion de Type 2: utiliser un algorithme qui repose sur l'adresse IP de l'appareil et réaliser une configuration explicite par l'intermédiaire de l'objet d'interface TCP/IP. Les appareils de Type 2 doivent mettre en œuvre la méthode algorithmique par défaut. Les appareils sont également fortement invités à mettre en œuvre la méthode de configuration explicite des adresses de multidiffusion. Les deux méthodes sont décrites ci-dessous.

11.2.4.9.5.2 Algorithme d'allocation fondé sur l'adresse IP de l'appareil

La plage des adresses de multidiffusion IP doit être l'Organizational Local Scope (la portée locale au sein d'une organisation) et doit commencer à 239.192.1.0. Chaque appareil doit

utiliser un bloc de 32 (au maximum) adresses de multidiffusion issues de cette plage. Chaque appareil doit calculer le bloc d'adresses de multidiffusion par un algorithme, décrit en plus de détails ci-dessous, qui utilise la partie Host Id (Identificateur d'hôte) de l'adresse IP de l'appareil.

Un Host Id d'appareil doit être déterminé en appliquant le masque de sous-réseau à l'adresse IP de l'appareil. Si un masque de sous-réseau est configuré pour l'appareil, le masque de sous-réseau doit être appliqué à l'adresse IP pour déterminer le Host Id. Si aucun masque de sous-réseau n'est utilisé, la classe de l'adresse IP de l'appareil doit être utilisée le Host Id (par exemple, pour les adresses de Classe C, 8 bits de Host Id doivent être utilisés; pour les adresses de Classe B, 16 bits de Host Id doivent être utilisés; et pour les adresses de Classe A, 24 bits de Host Id doivent être utilisés).

Afin de maintenir les adresses de multidiffusion IP dans les limites de l'Organization Local Scope de l'IPv4 et imposer des bornes raisonnables au nombre d'adresses de multidiffusion utilisées, les appareils doivent utiliser au maximum les 10 bits de poids faible du Host Id pour générer la plage d'adresses de multidiffusion. Cela permet 1 024 Host Id uniques.

Le pseudocode ci-après montre l'algorithme pour déterminer les adresses de multidiffusion de début et de fin de l'appareil:

```
Type2_Mcast_Base_Addr = 0xEFC00100 // 239.192.1.0 is the starting address
Type2_Host_Mask = 0x3FF           // 10 bits of host id

if Subnet_mask configured then
   Netmask = Subnet_mask
else
   if IP_address is Class A then
      Netmask = 255.0.0.0
   else if IP_address is Class B then
      Netmask = 255.255.0.0
   else if IP_address is Class C then
      Netmask = 255.255.255.0
end_else

Host_id = IP_addr & (~Netmask)
Mcast_index = Host_id - 1
Mcast_index = Mcast_index & (Type2_Host_Mask)

Mcast_start_addr = Type2_Mcast_Base_Addr + (Mcast_index * 32)
Mcast_end_addr = Mcast_start_addr + 31
```

Le Tableau 295 montre des exemples d'attributions de multidiffusion.

Tableau 295 – Exemple d'attributions de multidiffusion

Masque de sous-réseau	Adresse IP	Adresses de multidiffusion
Aucun configuré (8 bits de Host Id utilisés, car 192.168.x.x est de Classe C)	192.168.1.1	239.192.1.0 à 239.192.1.31
	192.168.1.2	239.192.1.32 à 239.192.1.63
	192.168.1.3	239.192.1.64 à 239.192.1.95
255.255.248.0 (11 bits de Host Id)	10.10.16.1	239.192.1.0 à 239.192.1.31
	10.10.16.2	239.192.1.32 à 239.192.1.63
	10.10.16.3	239.192.1.64 à 239.192.1.95
	10.10.20.0 (Host Id = 1 024; 10 bits de poids faible tous à 0)	239.192.128.224 à 239.192.128.255 (c'est la plage d'adresses de multidiffusion la plus élevée qui résulterait de l'utilisation de l'algorithme)

Le nombre des Host Id uniques étant fini, il est possible pour des appareils différents de produire des données utilisant la même adresse de multidiffusion. En conséquence, les appareils qui reçoivent des paquets sur des connexions de multidiffusion de Type 2 doivent filtrer les paquets entrants selon l'ID de connexion ainsi que selon l'adresse IP de l'appareil expéditeur. Ceci est décrit en 11.3.4.

Lorsque l'adresse IP de l'appareil ou le masque de sous-réseau change, les adresses de multidiffusion IP générées par l'algorithme doivent changer en conséquence.

11.2.4.9.5.3 Configuration explicite des adresses de multidiffusion IP

Les adresses de multidiffusion IP sont configurées par le biais de l'objet Interface TCP/IP (classe 0xF5). L'utilisateur (ou le logiciel) peut configurer une adresse de multidiffusion de début et un nombre d'adresses de multidiffusion à allouer. Les adresses de multidiffusion configurées doivent alors être utilisées pour les paquets de multidiffusion de Type 2.

Les appareils de Type 2 doivent au moins mettre en œuvre la méthode algorithmique pour allouer les adresses de multidiffusion IP et sont invités à mettre en œuvre les deux méthodes. Si les deux méthodes sont mises en œuvre, la méthode algorithmique doit être la méthode "out of box" par défaut.

11.3 Mise en correspondance des PDU des classes de transport 0 et 1

11.3.1 Datagrammes UDP

Les PDU pour les connexions de classe de transport 0 et 1 doivent être transmises à l'aide de l'UDP. Les PDU pour les connexions multipoints doivent être émises en utilisant la multidiffusion IP.

Le paquet doit être formaté comme cela est montré dans le Tableau 296. Notez que les paquets utilisent le format commun de paquet (Common Packet Format) défini en 4.3.3, mais sans l'en-tête d'encapsulation.

Tableau 296 – Format des données UDP pour la classe 0 et la classe 1

Nom de champ	Type	Valeur
Item Count	UINT	2
Type ID	UINT	0x8002 (Type d'adresse en séquence)
Length	UINT	8
Address Data	UDINT	ID de connexion (issu de la réponse Forward_open)
	UDINT	Numéro de séquence d'encapsulation. Il est différent du compte de séquence dans le paquet de classe de transport 1 (voir 11.3.3)
Type ID	UINT	0x00B1 (Type de données connectées)
Length	UINT	Nombre d'octets dans la PDU à suivre
Données	ARRAY of octets	PDU de la classe de transport 0 ou 1

11.3.2 Aucune liaison avec des connexions TCP

Afin d'ouvrir une connexion de classe de transport 0 ou 1, une connexion TCP et une session d'encapsulation doivent d'abord être établies. La connexion TCP est utilisée pour envoyer la demande Forward_Open et recevoir la réponse Forward_Open. Une fois la connexion TCP ouverte et les connexions de classe de transport 0 ou 1 établies, il est recommandé que les appareils de Type 2 laissent ouverte la connexion TCP. Si la connexion TCP est laissée ouverte, elle est alors disponible pour des communications ultérieures telles qu'une Forward_Close ou autres messages explicites.

Bien qu'il soit recommandé que les appareils laissent ouverte la connexion TCP, il ne doit y avoir aucune liaison entre la connexion TCP utilisée pour ouvrir la connexion de classe de transport 0 ou 1 et la connexion obtenue de classe 0 ou de classe 1. Si une connexion TCP se ferme, la fermeture de la connexion TCP ne doit pas amener la cible ou l'émetteur à fermer une quelconque connexion correspondante de classe de transport 0 ou 1.

11.3.3 Ordre des paquets de classe 0 et de classe 1

NOTE 1 Par définition, les transports de classe 0 et de classe 1 ne détectent pas les paquets hors ordre de classement. Pour la classe 0, chaque paquet est considéré comme étant une nouvelle donnée. Pour la classe 1, seuls les doublons de données sont détectés. Un paquet reçu est considéré comme une nouvelle donnée lorsque le compte de séquence est différent (supérieur ou inférieur) du compte de séquence du paquet précédent).

NOTE 2 Lorsque l'UDP est utilisé pour transporter des données connectées de classe 0 ou de classe 1, il n'y a aucune garantie que les paquets arrivent dans le même ordre suivant lequel ils ont été envoyés. Lorsque l'expéditeur et le destinataire se trouvent tous les deux dans le même sous-réseau, les paquets arrivent typiquement dans l'ordre. Cependant, lorsqu'ils passent par des routeurs, s'il existe plusieurs chemins qu'un paquet pourrait emprunter, il est possible que des paquets arrivent hors ordre de classement.

Pour les connexions de classe 0 et de classe 1 sur Ethernet, les appareils doivent maintenir un numéro de séquence dans la partie "données" du diagramme UDP défini en 11.3.1. Le numéro de séquence d'encapsulation doit être maintenu par connexion. Chaque fois qu'un appareil Ethernet envoie un paquet de Classe 1, il doit incrémenter le numéro de séquence pour la connexion en question. Si l'appareil Ethernet destinataire reçoit un paquet dont le numéro de séquence est inférieur à celui du paquet reçu précédemment, le paquet ayant le numéro de séquence plus petit doit être rejeté.

Le numéro de séquence doit être soumis à une opération avec une arithmétique modulaire pour traiter du repassage par zéro de la séquence.

NOTE 3 Le traitement des numéros de séquence de 32 bits est décrit dans l'IETF RFC 793 (la définition de TCP).

11.3.4 Filtrage des données connectées entrantes

Les appareils Ethernet qui reçoivent des données de classe 0 et de classe 1 doivent filtrer les paquets entrants selon l'ID de connexion et l'adresse IP de l'appareil expéditeur. Cela est nécessaire pour les raisons suivantes:

- Pour les connexions multipoints, il n'existe pas de mécanisme garanti permettant d'empêcher plusieurs appareils d'utiliser la même adresse de multidiffusion IP. En conséquence, un appareil pourrait recevoir des données de multidiffusion (trompeuses) provenant d'un appareil avec lequel il n'a pas établi de connexion.
- Pour les connexions de multidiffusion, un appareil est autorisé à utiliser la même adresse de multidiffusion IP pour plusieurs connexions multipoints de classe 0 et de classe 1.
- Prévenir les conflits d'ID de connexion.

Lorsqu'une connexion de classe 0 ou de classe 1 est établie, les appareils Ethernet cibles et émetteurs doivent enregistrer l'ID de connexion sur lequel ils recevront des données connectées, couplé avec l'adresse IP de l'appareil à l'autre extrémité de la connexion. Lorsqu'un appareil reçoit des données connectées, il doit confirmer que l'ID de connexion est valide pour l'adresse IP de l'appareil expéditeur. Si tel n'est pas le cas, le paquet doit être rejeté.

11.4 Mise en correspondance des PDU des classes de transport 2 et 3

Les PDU pour les connexions de classes de transport 2 et 3 doivent être émises sur une connexion TCP en utilisant le protocole d'encapsulation défini en 11.8 et en 4.3.1 (commande SendUnitData).

Les données connectées doivent être formatées comme le montre le Tableau 297.

Tableau 297 – Données connectées des classes de transport 2 et 3

Structure	Nom de champ		Type de données	Valeur du champ
En-tête d'encapsulation	Command		UINT	SendUnitData (0x70)
	Length		UINT	Longueur d'une portion de données spécifique à la commande
	Session handle		UDINT	Identification retournée par RegisterSession
	Status		UDINT	0
	Sender Context		ARRAY of 8 USHORT	Toute valeur (ignoré par la cible)
	Options		UDINT	0
Données spécifiques à la commande	Interface handle		UDINT	0
	Timeout		UINT	Toute valeur (ignoré par la cible)
	Encapsulated packet (dans le format de paquet normalisé)	Item count	UINT	2 (indique 1 élément adresse, 1 élément données)
		Address Type ID	UINT	0x00A1 (élément Connected Address)
		Address length	UINT	4
		Address Data	UDINT	Connexion ID from Forward_Open/ Forward_Open_Reply
		Data Type ID	UINT	0x00B1 (Elément Connected Data)

Structure	Nom de champ		Type de données	Valeur du champ
		Data length	UINT	Longueur du champ suivant en octets (c'est-à-dire longueur de la PDU des classes de transport 2/3, y compris le compte de séquence)
		Données	ARRAY of USINT	PDU des classes de transport 2/3 (y compris le compte de séquence)

Plusieurs connexions de Type 2 peuvent être envoyées sur une seule connexion TCP. Une mise en œuvre peut ne pas prendre en charge un nombre spécifique de connexions par connexion TCP. Une mise en œuvre peut imposer une borne supérieure si elle le veut.

En raison de la nature duplex intégral du protocole TCP, les connexions de liaisons O2T et T2O doivent utiliser la même connexion TCP. Cependant si une cible crée ultérieurement une connexion de Type 2, elle doit être considérée comme un émetteur et une connexion TCP différente doit être utilisée.

NOTE La présente norme ne définit pas d'exigences pour la gestion de la connexion TCP, telles que les temporisations d'inactivité ou la fermeture de la connexion TCP lorsque toutes les connexions natives sont fermées. Les mises en œuvre sont toutefois libres de les mettre en œuvre.

Les cibles et les émetteurs doivent fermer toutes les connexions de Type 2 de classe de transport 3 ou 3 lorsque la connexion TCP émettrice correspondante est fermée.

11.5 Mise en correspondance des classes de transport 4 à 6

Le protocole d'encapsulation défini en 11.8 et en 4.3.1 ne doit pas être utilisé pour encapsuler des PDU pour les classes de transport 4, 5 et 6.

11.6 Utilisation de l'IGMP

11.6.1 Contexte (informative)

Le protocole de gestion de groupe internet (IGMP) est un protocole normalisé utilisé par des hôtes pour rapporter leurs adhésions comme membres du groupe de multidiffusion IP et doit être mis en œuvre par tout hôte qui souhaite recevoir des datagrammes de multidiffusion IP. Les messages IGMP sont utilisés par des routeurs de multidiffusion pour savoir quels groupes de groupe de multidiffusion ont des membres sur leurs réseaux rattachés. Les messages IGMP sont également utilisés par des commutateurs capables de prendre en charge l'écoute IGMP ("IGMP snooping") qui permet au commutateur d'écouter les messages IGMP et d'envoyer uniquement les paquets de multidiffusion vers les ports qui ont rejoint le groupe de multidiffusion.

Il existe deux versions de l'IGMP:

- La version IGMP V1 est définie dans l'IETF RFC 1112.
- La version IGMP V2 est définie dans l'IETF RFC 2236.
- La version IGMP V3 est définie dans l'IETF RFC 3376.

L'IETF RFC 2236 et l'IETF RFC 3376 évoquent les exigences en termes d'hôte et de routeur pour une interopération avec les anciennes versions de l'IGMP.

Comme les appareils de Type 2 font une utilisation extensive de la multidiffusion IP pour les connexions des classes de transport 0 et 1, une utilisation cohérente de l'IGMP par les appareils de Type 2 est essentielle pour créer des réseaux d'application de Type 2 qui fonctionnent correctement.

11.6.2 Messages de Membership Report (rapports sur les adhésions comme membres) de l'IGMP

Les appareils de Type 2 doivent émettre un message Membership Report lorsqu'ils ouvrent une connexion de Type 2 sur laquelle ils recevront des paquets de multidiffusion. Plus précisément, les appareils doivent respecter le comportement ci-après:

- Lorsque le type de connexion T->O est multipoint (l'émetteur est un consommateur multipoint), l'émetteur doit produire un Membership Report à la réception d'une réponse Forward_Open réussie. Le Membership Report doit inclure l'adresse de multidiffusion IP telle que communiquée dans la réponse Forward_Open.
- Lorsque le type de connexion O->T est multipoint (la cible est un consommateur multipoint), la cible doit produire un Membership Report à l'envoi d'une réponse Forward_Open réussie. Le Membership Report doit inclure l'adresse de multidiffusion IP telle que communiquée dans la Forward_Open.

Si l'appareil a déjà émis un Membership Report pour l'adresse de multidiffusion IP (par exemple, si l'adresse de multidiffusion est utilisée avec une connexion existante), l'appareil peut produire un autre Membership Report, mais il n'est pas tenu de le faire.

Les appareils doivent également envoyer des messages Membership Report en réponse aux messages Membership Query (Interrogation sur les adhésions comme membres), conformément aux RFC de l'IGMP.

11.6.3 Messages "Leave Group" de l'IGMP

Les appareils qui prennent en charge l'IGMP V2 doivent émettre un Leave Group ("quitter le groupe") lorsque toutes les connexions de Type 2 associées à une adresse de multidiffusion IP consommatrice ont été soit fermées, soit temporisées. Plus précisément, les appareils doivent respecter le comportement ci-après:

- Lorsque le type de connexion T->O est multipoint (l'émetteur est un consommateur multipoint), l'émetteur doit produire un Leave Group à la réception d'une réponse Forward_Close réussie si l'émetteur n'a pas d'autres connexions ouvertes consommant sur l'adresse de multidiffusion IP en question.
- Lorsque le type de connexion O->T est multipoint (la cible est un consommateur multipoint), la cible doit produire un Leave Group à l'envoi d'une réponse Forward_Close réussie si la cible n'a pas d'autres connexions ouvertes consommant sur l'adresse de multidiffusion IP en question.
- En cas de temporisation d'une connexion, le consommateur multipoint (qu'il soit cible ou émetteur) doit produire un message Leave Group si le consommateur multipoint n'a pas d'autres connexions consommant sur l'adresse de multidiffusion IP en question.

11.7 Qualité de service (QoS) pour les messages CP 2/2

11.7.1 Vue d'ensemble

"Qualité de service" (QoS) est un terme général pour les mécanismes qui traitent les flux de trafic avec des priorités relatives différentes ou autres caractéristiques de livraison. Les mécanismes QoS normalisés comprennent l'IEEE 802.1D/IEEE 802.1Q (Priorité des trames Ethernet) et les services différenciés (Differentiated Services, DiffServ) dans la suite de protocoles TCP/IP.

Dans le contexte des applications CPF 2 et CP 2/2, la qualité de service est spécialement importante pour les applications à temps critique dans lesquelles la livraison des paquets affectera la stabilité de l'application.

CP 2/2 définit des exigences et des recommandations relatives à la manière dont les appareils CP 2/2 utilisent deux mécanismes QoS normalisés: IEEE 802.1D/IEEE 802.1Q and

Differentiated Services. Les mécanismes QoS définis pour les appareils CP 2/2 sont résumés ci-après.

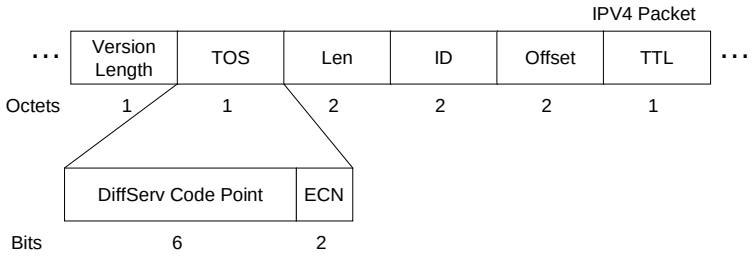
- L'approche globale est que les appareils finals CP 2/2 marquent les paquets CP 2/2 avec des valeurs de priorité, en utilisant les mécanismes normalisés susmentionnés. Le marquage des paquets avec une priorité permet aux appareils d'infrastructure (par exemple, les commutateurs et les routeurs) de mieux différencier le trafic CP 2/2.
- Pour les connexions de classes de transport 0 et 1 CPF 2 (à savoir fondées sur UDP), il existe une mise en correspondance définie des priorités CPF 2 aux priorités de l'IEEE 802.1D et aux points de code DiffServ (voir 11.7.4).
- Pour les connexions de l'UCMM et les connexions 2 et 3 de la classe de transport CPF 2 (c'est-à-dire dérivées du protocole TCP) et tous les autres messages d'encapsulation CP 2/2 (dérivés soit de TCP soit d'UDP), le DiffServ Code Point est défini et la valeur de priorité IEEE 802.1D est définie.
- Pour les messages PTP (CEI 61588), il y a des points de code DiffServ Code Points et des valeurs de priorité de l'IEEE 802.1D correspondant aux deux différents types de message PTP.
- Il est fortement recommandé que les appareils marquent par défaut les messages CPF 2 et PTP avec des valeurs de DSCP (point de code de service différencié).
- En plus des DSCP, les appareils peuvent facultativement prendre en charge l'envoi et la réception de trames étiquetées IEEE 802.1Q. S'il est pris en charge, l'envoi de trames étiquetées doit être désactivé par défaut afin de prévenir les problèmes d'interopérabilité des appareils. Lorsque l'étiquetage IEEE 802.1Q est souhaité, l'utilisateur final doit activer explicitement l'envoi de trames IEEE 802.1Q et doit assurer que les appareils tant expéditeurs que destinataires prennent en charge les trames étiquetées.
- L'objet QoS fournit un moyen de configurer les valeurs de DSCP et aussi un moyen d'activer/désactiver l'envoi de trames étiquetées IEEE 802.1Q (voir la CEI 61158-4-2, 7.10).
- Il n'y a aucune exigence que les appareils marquent le trafic autre que CPF 2 ou CEI 61588. Les appareils peuvent le faire, en fonction des capacités de leurs mises en œuvre.
- A l'heure actuelle, il n'y aucune exigence de mises en œuvre spécifiques pour que les appareils finals différencient en interne le trafic avec des priorités différentes. La différenciation du trafic dans les appareils finals est un domaine de développement futur. Il est recommandé au minimum aux mises en œuvre de CP 2/2 d'accorder une plus haute priorité au traitement des messages implicites de CP 2/2 qu'aux messages explicites. Une différenciation plus poussée, basée sur les mécanismes QoS ci-dessus, dans la mesure du possible, est également recommandée.

Les références suivantes sont applicables à la qualité de service pour les appareils CP 2/2: IEEE 802.1D, IEEE 802.1Q, IETF RFC 2474, IETF RFC 2475, IETF RFC 2597, IETF RFC 2873, IETF RFC 3140, IETF RFC 3246, IETF RFC 4594.

11.7.2 Format des DSCP

Les services différenciés (DiffServ) sont un modèle pour spécifier la priorité relative de trafic basée sur le champ "type de service" (type of service, ToS) d'un paquet IPv4. Le modèle est défini dans l'IETF RFC 2475. DiffServ permet à des nœuds d'acheminer des paquets en se basant sur la classe de trafic telle que définie par le point de code DiffServ (DSCP, Codepoint DiffServ) et les caractéristiques de comportement par saut (PHB, Per-Hop Behavior).

La Figure 60 montre le champ DS dans l'en-tête IP.



Légende

Anglais	Français
Version length	Longueur de version
DiffServ Code Point	Point de code de service différencié

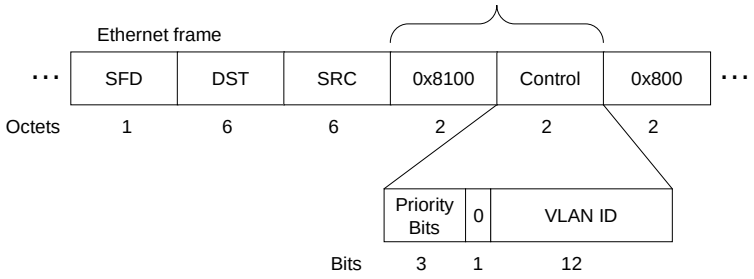
Figure 60 – Champ DS dans l'en-tête IP

Lors du positionnement de la valeur du DSCP dans l'en-tête IP, si elle est placée directement dans le champ ToS, elle doit être décalée de 2 bits vers la gauche.

11.7.3 Format IEEE 802.1D/Q

L'IEEE 802.1Q définit un format de trames Ethernet qui permet d'inclure l'ID de VLAN et une priorité. La trame IEEE 802.1Q a un EtherType de 0x8100 et un préfixe de 4 octets entre les champs Source et Type de la trame. La trame étiquetée définit un champ de 3 bits pour spécifier 8 niveaux de priorité, qui sont spécifiés plus en détails dans l'IEEE 802.1D. La priorité 7 est la plus élevée. La priorité 0 est la plus basse.

La Figure 61 montre le format d'une trame IEEE 802.1Q.



Légende

Anglais	Français
Ethernet frame	Trame Ethernet
Priority bits	Bits de priorité
VLAN ID	Identificateur de VLAN

Figure 61 – Trame étiquetée IEEE 802.1Q

11.7.4 Mise en correspondance du trafic CPF 2 avec DSCP et IEEE 802.1D

Le Tableau 298 définit les mises en correspondance des DSCP par défaut et des priorités IEEE 802.1D pour le trafic CP 2/2 et CP 2/2.1 (CEI 61588).

Tableau 298 – Mise en correspondance des DSCP par défaut et de l'IEEE 802.1D

Type de trafic	Priorité CPF 2	DSCP	Priorité IEEE 802.1D[a]	Utilisation de trafic CPF 2 (recommandée)
Evénement PTP (CEI 61588)	n/a	59 ('111011')	7	n/a
PTP General (CEI 61588)	n/a	47 ('101111')	5	n/a
Classe 0/1 CPF 2	Urgent (3)	55 ('110111')	6	Mouvement CIP
	Scheduled (2)	47 ('101111')	5	E/S de sécurité, E/S
	High (1)	43 ('101011')	5	E/S (I/O)
	Low (0)	31 ('011111')	3	Aucune recommandation à l'heure actuelle
UCMM de CPF 2 CPF 2 de classe 2/3 Tous les autres messages d'encapsulation CP 2/2	Toutes	27 ('011011')	3	Messagerie CPF 2

[a] L'envoi de trames étiquetées IEEE 802.1Q est désactivé par défaut.

NOTE 1 Dans le Tableau 298, les valeurs IEEE 802.1D et DSCP pour les messages des classes de transport 0 et 1 sont basées sur la priorité CPF 2 (indiqué dans le service Forward_Open). Les messages explicites PTP et CPF 2 utilisent des valeurs IEEE 802.1D et DSCP indépendantes de la priorité CPF 2.

Les valeurs par défaut de DSCP et l'activation/désactivation de trames étiquetées de l'IEEE 802.1Q peuvent être changées par l'intermédiaire de l'Objet QoS (voir CEI 61158-4-2, 7.10).

NOTE 2 Les valeurs par défaut de DSCP sont des valeurs "local use" (utilisation locale), telles que définies dans l'IETF RFC 2474.

11.7.5 Utilisation CP 2/2 des DSCP

Lorsqu'ils marquent le trafic CP 2/2 avec des valeurs de DSCP, les appareils CP 2/2 doivent utiliser les valeurs telles que configurées dans l'objet QoS (les valeurs par défaut sont montrées dans le Tableau 298). L'utilisation du champ ECN dans l'en-tête IP par des appareils finals CP 2/2 n'est pas définie actuellement et le champ doit être mis à 0.

A remarquer que quand le marquage DSCP est pris en charge, tous les paquets sortants pour les connexions établies d'encapsulation TCP doivent être marqués de la valeur appropriée DSCP, y compris les paquets qui ne comportent que l'acquittement. Il est recommandé que tous les paquets sortants qui interviennent à l'ouverture et à la clôture de ces connexions soient aussi marqués (SYN, FIN, RST et les acquittements associés).

Les appareils sont fortement invités à prendre en charge le marquage des paquets CP 2/2 avec les valeurs de DSCP. Lorsque le marquage est pris en charge, le comportement par défaut doit être de marquer les paquets.

Tous les appareils CP 2/2 doivent prendre en charge les paquets destinataires avec des valeurs de DSCP non nulles car il s'agit d'une exigence du Protocole Internet (voir IETF RFC 791 et IETF RFC 1122). Des appareils d'infrastructure Ethernet (par exemple, les commutateurs et les routeurs) peuvent potentiellement altérer les valeurs de DSCP. Les appareils destinataires ne doivent pas présupposer ou autrement vérifier que les valeurs de DSCP entrantes sont les mêmes que dans le Tableau 298 ou sont les mêmes valeurs envoyées par l'appareil expéditeur.

NOTE Ce comportement est cohérent avec les modifications apportées à TCP (IETF RFC 793) comme le décrit IETF RFC 2873.

11.7.6 Utilisation CP 2/2 de l'IEEE 802.1D/Q

Lors de l'envoi de trafic avec des trames étiquetées de l'IEEE 802.1Q, les appareils CP 2/2 doivent utiliser les valeurs de priorité spécifiées dans le Tableau 298. L'ID de VLAN doit être mis à 0, à moins qu'un ID de VLAN spécifique pour l'appareil n'ait été configuré par un moyen qui n'est pas couvert par la présente spécification. Lors de la réception de trames étiquetées de l'IEEE 802.1Q, les appareils ne doivent pas exiger que l'ID de VLAN soit mis à 0.

Lors de l'envoi de trames étiquetées de l'IEEE 802.1Q, les appareils doivent également régler la valeur de DSCP correspondante dans l'en-tête IP.

Lorsque les trames IEEE 802.1Q sont prises en charge, l'appareil doit aussi prendre en charge l'objet QoS (voir CEI 61158-4-2, 7.10). Le comportement par défaut doit être de désactiver l'envoi de trames étiquetées de l'IEEE 802.1Q, car l'envoi de trames étiquetées par défaut peut conduire à des problèmes d'interopérabilité des appareils.

A remarquer que quand le trafic est envoyé à l'aide de trames étiquetées de l'IEEE 802.1Q, tous les paquets sortants destinés aux connexions établies d'encapsulation TCP doivent utiliser la valeur de priorité de l'IEEE 802.1D spécifiée, y compris les paquets qui ne comportent que l'acquittement. Il est recommandé que tous les paquets sortants qui interviennent à l'ouverture et à la clôture de ces connexions (SYN, FIN, RST et les acquittements associés) utilisent aussi la même valeur de priorité de l'IEEE 802.1D.

11.7.7 Considérations utilisateur avec l'IEEE 802.1D/Q

Certains appareils CP 2/2 ne supportent pas la réception de trames étiquetées de l'IEEE 802.1Q. Lorsque des trames de l'IEEE 802.1Q sont envoyées à ces appareils, elles seront rejetées. En outre, certains commutateurs gérés rejetteront les trames étiquetées de l'IEEE 802.1Q, à moins que le port de réception ne soit configuré pour les accepter.

Afin de prévenir les problèmes d'interopérabilité, l'envoi de trames IEEE 802.1Q est désactivé par défaut pour les appareils CP 2/2. L'utilisateur final peut choisir d'activer l'envoi des trames étiquetées lorsqu'elles sont prises en charge, par l'intermédiaire de l'objet QoS. L'utilisateur est responsable en dernier ressort de s'assurer que les appareils tant expéditeurs que destinataire prennent en charge les trames IEEE 802.1Q et que l'infrastructure du réseau est correctement configurée.

11.8 Gestion d'une session d'encapsulation

11.8.1 Phases d'une session d'encapsulation

Une session d'encapsulation doit comporter trois phases:

- établissement d'une session;
- maintien d'une session;
- fermeture d'une session.

11.8.2 Etablissement d'une session

L'établissement d'une session doit se poursuivre selon les étapes suivantes:

- l'émetteur doit ouvrir une connexion TCP/IP vers la cible, en utilisant le numéro de port réservé (0xAF12) ou, si spécifié, le numéro de port issu du chemin de connexion;
- l'émetteur doit envoyer une commande RegisterSession à la cible (voir 4.3.2.2);
- la cible doit vérifier la version de protocole avec le message de commande pour vérifier qu'elle prend en charge la même version de protocole que l'émetteur. Si tel n'est pas le cas, la cible doit retourner une RegisterSession avec un champ Status approprié, avec la version la plus élevée du protocole pris en charge;

- la cible doit vérifier les fanions d'option dans la commande pour s'assurer que les options requises sont prises en charge. Si tel n'est pas le cas, la cible doit retourner une réponse RegisterSession avec un champ Status approprié, ainsi que les options prises en charge;
- la cible doit assigner un nouvel ID de session (unique) et doit envoyer une réponse RegisterSession à l'émetteur

Les émetteurs ne doivent pas enregistrer plus d'une session active sur une même connexion TCP.

NOTE Un numéro de port TCP, 0xAF12, a été réservé par l'Internet Assigned Numbers Authority (IANA) pour une utilisation par le protocole d'encapsulation.

11.8.3 Arrêt d'une session

L'émetteur ou la cible peut mettre fin à la session. Les sessions doivent être arrêtées de l'une de deux manières:

- l'émetteur ou la cible doit fermer la connexion TCP sous-jacente. La cible ou l'émetteur correspondant(e) doit détecter la perte de la connexion TCP et doit fermer son côté de la connexion;
- l'émetteur ou la cible doit envoyer une commande UnRegisterSession (voir 4.3.2.3) et doit attendre de détecter la fermeture de la connexion TCP La cible ou l'émetteur correspondant(e) doit ensuite fermer son côté de la connexion TCP. L'expéditeur de l'UnRegisterSession doit détecter la perte de la connexion TCP puis il doit fermer son côté de la connexion;

NOTE La seconde méthode est préférentielle parce qu'elle conduit à une mise au net plus opportune de la connexion TCP.

11.8.4 Maintien d'une session

Une fois une session établie, elle doit rester établie jusqu'à ce que l'une des situations suivantes se produise:

- l'émetteur ou la cible ferme la connexion TCP;
- l'émetteur ou la cible produit la commande UnRegisterSession;
- la connexion TCP est rompue.

12 Diagramme protocolaire de mise en correspondance DLL 3 (DMPM 3)

Ce DMPM est spécifié dans la CEI 62026-3:2008, Article 5.

Bibliographie

NOTE Toutes les parties de la série CEI 61158, ainsi que la CEI 61784-1 et la CEI 61784-2 font l'objet d'une maintenance simultanée. Les références croisées à ces documents dans le texte se rapportent donc aux éditions datées figurant dans cette liste de références bibliographiques.

CEI 61131-3, *Automates programmables – Partie 3: Langages de programmation*

CEI 61784-1:2014, *Réseaux de communication industriels – Profils – Partie 1: Profils de bus de terrain*

CEI 61784-2:2014, *Réseaux de communication industriels – Profils – Partie 2: Profils de bus de terrain supplémentaires pour les réseaux en temps réel basés sur l'ISO/CEI 8802-3*

ISO/CEI 8822, *Technologies de l'information – Interconnexion de systèmes ouverts – Définition du service de présentation*

ISO/IEC/IEEE 60559, *Information technology – Microprocessor Systems – Floating-Point arithmetic* (disponible en anglais seulement)

IETF RFC 768, *User Datagram Protocol*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 793, *Transmission Control Protocol*, disponible à l'adresse <http://www.ietf.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 1: Common Industrial Protocol (CIPTM) – Edition 3.13, November 2012*, disponible à l'adresse <http://www.odva.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 2: EtherNet/IPTM Adaptation of CIP – Edition 1.14, November 2012*, disponible à l'adresse <http://www.odva.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 3: DeviceNetTM Adaptation of CIP – Edition 1.12, November 2011*, disponible à l'adresse <http://www.odva.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 4: ControlNetTM Adaptation of CIP – Edition 1.7, April 2011,* disponible à l'adresse <http://www.odva.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 5: CIP SafetyTM – Edition 2.6, November 2012*, disponible à l'adresse <http://www.odva.org>

———————

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