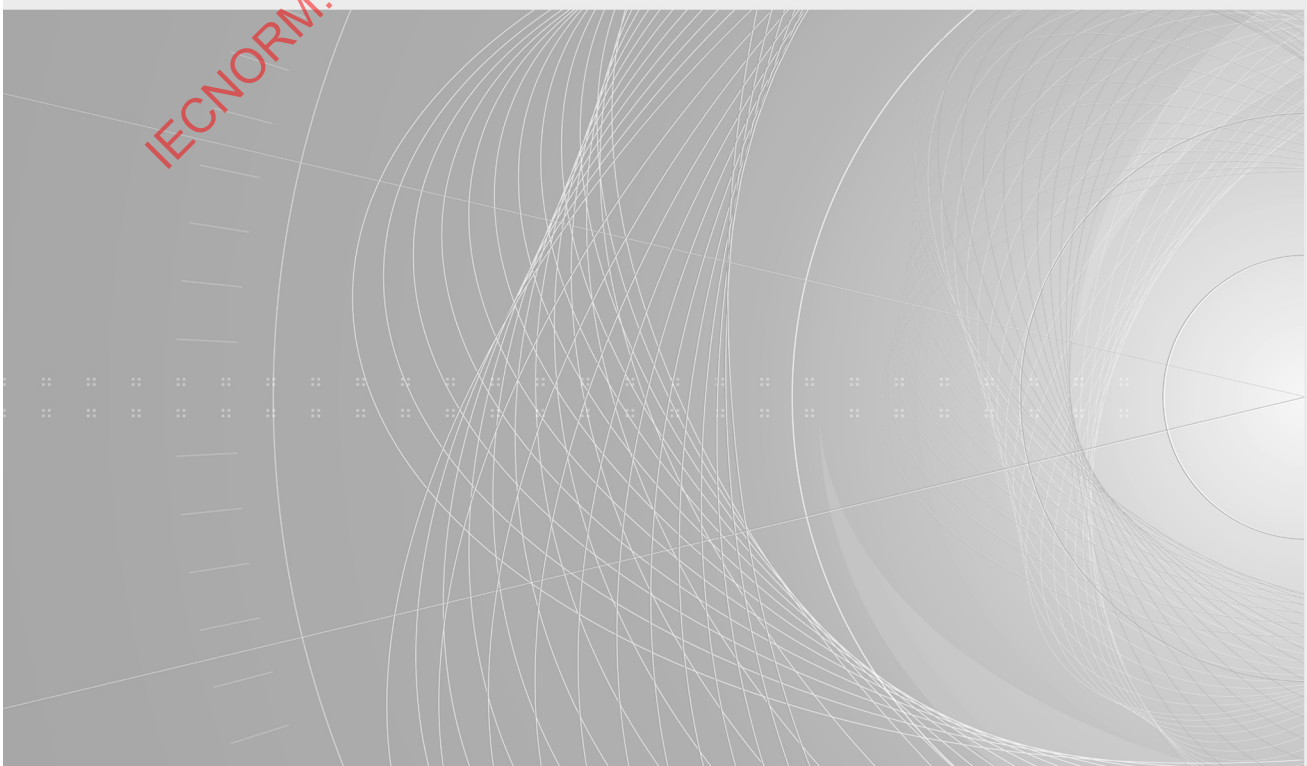
IEC 61158-6-26

Edition 1.0 2019-06

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications –
Part 6-26: Application layer protocol specification – Type 26 elements



IEC 61158-6-26:2019-06(en)

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

Tel.: +41 22 919 02 11
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 corrigendum or an amendment might have been published.

IEC publications search - webstore.iec.ch/advsearchform
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 It also gives information on projects, replaced and withdrawn publications.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 once a month by email.

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: sales@iec.ch.

Electropedia - www.electropedia.org
The world's leading online dictionary on electrotechnology, containing more than 22 000 terminological entries in English and French, with equivalent terms in 16 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary
67 000 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 61158-6-26

Edition 1.0 2019-06

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications – Part 6-26: Application layer protocol specification – Type 26 elements

INTERNATIONAL ELECTROTECHNICAL COMMISSION

ICS 25.040.40; 35.100.70; 35.110

ISBN 978-2-8322-7016-5

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-26: Application layer protocol specification – Type 26 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.

International standard IEC 61158-6-26 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement and control.

The text of this International Standard is based on the following documents:

FDIS	Report on voting
65C/948/FDIS	65C/956/RVD

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

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

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

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

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

A bilingual version of this publication may be issued at a later date.

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 document 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 document is concerned, in particular, with the communication and interworking of sensors, effectors and other automation devices. By using this document 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 6-26: Application layer protocol specification – Type 26 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 part of IEC 61158 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 26 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 International Standard defines in an abstract way the externally visible behavior provided by the Type 26 of the fieldbus Application Layer in terms of:

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

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

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

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

The purpose of this document is to define the protocol provided to:

a) define the wire-representation of the service primitives defined in IEC 61158-5-26, and

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

This document specifies the protocol of the Type 26 fieldbus Application Layer, in conformance with the OSI Basic Reference Model (see ISO/IEC 7498-1) and the OSI Application Layer Structure (see ISO/IEC 9545).

FAL services and protocols are provided by FAL application-entities (AE) contained within the application processes. The FAL AE is composed of a set of object-oriented Application Service Elements (ASEs) and a Layer Management Entity (LME) that manages the AE. The ASEs provide communication services that operate on a set of related application process object (APO) classes. One of the FAL ASEs is a management ASE that provides a common set of services for the management of the instances of FAL classes.

Although these services specify, from the perspective of applications, how request and responses are issued and delivered, they do not include a specification of what the requesting and responding applications are to do with them. That is, the behavioral aspects of the applications are not specified; only a definition of what requests and responses they can

send/receive is specified. This permits greater flexibility to the FAL users in standardizing such object behavior. In addition to these services, some supporting services are also defined in this document to provide access to the FAL to control certain aspects of its operation.

1.2 Specifications

The principal objective of this document is to specify the syntax and behavior of the application layer protocol that conveys the application layer services defined in IEC 61158-5-26.

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

1.3 Conformance

This document does not specify individual implementations or products, nor does it constrain the implementations of application layer entities within industrial automation systems.

There is no conformance of equipment to the application layer service definition standard. Instead, conformance is achieved through implementation of this application layer protocol specification.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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 61158-1:2019, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61158-5-26:2019, *Industrial communication networks – Fieldbus specification – Part 5-26: Application layer service definition – Type 26 elements*

IEC 61784-2:2019, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC/IEEE 8802-3*

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

ISO/IEC/IEEE 8802-3, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Standard for Ethernet*

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

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 9899, *Information technology – Programming languages – C*

IETF RFC 768, *User Datagram Protocol*, available at http://www.ietf.org [viewed 2018-09-20]

IETF RFC 791, *Internet Protocol*, available at http://www.ietf.org [viewed 2018-09-20]

IETF RFC 792, *Internet Control Message Protocol*, available at http://www.ietf.org [viewed 2018-09-20]

IETF RFC 793, *Transmission Control Protocol*, available at http://www.ietf.org [viewed 2018-09-20]

IETF RFC 796, *Address mappings,* available at http://www.ietf.org [viewed 2018-09-20]

IETF RFC 826, *An Ethernet Address Resolution Protocol*, available at http://www.ietf.org [viewed 2018-09-20]

IETF RFC 894, *A Standard for the Transmission of IP Datagrams over Ethernet Networks*, available at http://www.ietf.org [viewed 2018-09-20]

IETF RFC 919, *Broadcasting Internet Datagrams*, available at http://www.ietf.org [viewed 2018-09-20]

IETF RFC 922, *Broadcasting Internet Datagrams in the presence of subnets,* available at http://www.ietf.org [viewed 2018-09-20]

IETF RFC 950, *Internet Standard Subnetting Procedure*, available at http://www.ietf.org [viewed 2018-09-20]

3 Terms, definitions, symbols, abbreviations and conventions

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

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at http://www.electropedia.org/
- ISO Online browsing platform: available at http://www.iso.org/obp

3.1 Terms and definitions from other ISO/IEC standards

3.1.1 Terms and definitions from ISO/IEC 7498-1

For the purposes of this document, the following terms and definitions given in ISO/IEC 7498-1 apply.

a) application entity

b) application process

c) application protocol data unit
d) application service element
e) application entity invocation
f) application process invocation
g) application transaction
h) real open system
i) transfer syntax

3.1.2 Terms and definitions from ISO/IEC 8822

For the purposes of this document, the following terms and definitions given in ISO/IEC 8822 apply:

a) abstract syntax
b) presentation context

3.1.3 Terms and definitions from ISO/IEC 9545

For the purposes of this document, the following terms and definitions given in ISO/IEC 9545 apply:

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

3.1.4 Terms and definitions from ISO/IEC 8824-1

For the purposes of this document, the following terms and definitions given in ISO/IEC 8824-1 apply:

a) object identifier
b) type
c) value
d) simple type
e) structured type
f) component type
g) tag
h) Boolean type
i) true
j) false
k) integer type
l) bitstring type
m) octetstring type
n) null type

o) sequence type

p) sequence of type

q) choice type

r) tagged type

s) any type

t) module

u) production

3.1.5 Terms and definitions from ISO/IEC 8825-1

For the purposes of this document, the following terms and definitions given in ISO/IEC 8825-1 apply:

a) encoding (of a data value)

b) data value

c) identifier octets (the singular form is used in this document)

d) length octet(s) (both singular and plural forms are used in this document)

e) contents octets

3.2 Type 26 specific terms and definitions

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

3.2.1
application relationship
cooperative association between two or more application-entity-invocations for the purpose of exchange of information and coordination of their joint operation.

Note 1 to entry: This relationship is activated either by the exchange of application-protocol-data-units or as a result of preconfiguration activities.

3.2.2
application process object
component of an application process that is identifiable and accessible through an FAL application relationship

3.2.3
application relationship endpoint
context and behavior of an application relationship as seen and maintained by one of the application processes involved in the application relationship

Note 1 to entry: Each application process involved in the application relationship maintains its own application relationship endpoint.

3.2.4
client
<object view> object which uses the services of another (server) object to perform a task

3.2.5
client
<communication view> initiator of a message to which a server react

3.2.6
common memory
virtual memory accessible with logically unique address used for the cyclic-data transmission

Note 1 to entry: It is composed of the memory-area-1 and -2 of the memory size 512 words and 8 192 words respectively and is shared with the Type 26 nodes in a Type 26 fieldbus network.

3.2.7
connection
logical binding between application objects that are within the same or different devices

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

3.2.8
cyclic-data frame
frame used for the Type 26 cyclic-data transmission

3.2.9
cyclic-data transmission
data transmission with which a Type 26 node sends out data on a pre-assigned area for the node on the common-memory, in broadcasting to all other Type 26 nodes every time the node obtains a token

3.2.10
device
physical hardware connected to the link

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

3.2.11
device profile
collection of device dependent information and functionality providing consistency between similar devices of the same device type

3.2.12
error
discrepancy between a computed, observed or measured value or condition and the specified or theoretically correct value or condition

3.2.13
event
instance of a change of conditions

3.2.14
FA-link header
header part of the APDU used for Type 26 data transmission

3.2.15
FA-link protocol
communication protocol for Type 26 fieldbus

Note 1 to entry: It defines the token management, the cyclic data transmission and the message data transmission, with which each Type 26 node can perform data transmission on an equal basis.

3.2.16
final-cyclic-data frame
cyclic-data frame sent out immediately before sending a token-frame out of a Type 26 node

3.2.17
frame
synonym for DLPDU

3.2.18
invocation
act of using a service or other resource of an application process

Note 1 to entry: Each invocation represents a separate thread of control that can be described by its context. Once the service completes, or use of the resource is released, the invocation ceases to exist. For service invocations, a service that has been initiated but not yet completed is referred to as an outstanding service invocation.

3.2.19
in-ring-startup state
state of a node attempting to participate in Type 26 fieldbus network in the valid-linking state

3.2.20
major-version
indication of the major-version number of the FA-link protocol

Note 1 to entry: The version number of the FA-link protocol is given as a combination of a major- and a minor-version number. The value of the major-version number is in the range of 1 to 15 (encoded with four bits) and corresponds to the digits before the decimal point (for example, the major-version number is 3 for the FA-link protocol Version 3.01.).

3.2.21
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.

3.2.22
message-data frame
frame used for the Type 26 message-data transmission

3.2.23
message-data transmission
sporadic data transmission with which a Type 26 node sends out the message data of request, response, information report, and notification upon event happened at the node and so forth, to other Type 26 nodes

3.2.24
minor-version
indication of the minor-version number of the FA-link protocol

Note 1 to entry: The version number of the FA-link protocol is given as a combination of a major- and a minor-version number. The value of the minor-version number is in the range of 1 to 15 (encoded with four bits) and corresponds to the digits after the decimal point (for example, the minor-version number is 1 for the FA-link protocol Version 3.01.).

3.2.25
network load measurement function
provision for monitoring and measurement of the communication load over a Type 26 fiedbus with a Type 26 node

3.2.26
network-startup state
state of a node attempting to participate in Type 26 fieldbus network not yet in the valid-linking state

3.2.27
participation-request-frame
frame used for a Type 26 node notifying of its participation-request in a Type 26 fieldbus network

3.2.28
protocol type
identification of communication protocol

3.2.29
publisher
role of an AR endpoint that transmits APDUs onto the fieldbus for consumption by one or more subscribers

3.2.30
resource
processing or information capability of a subsystem

3.2.31
subscriber
role of an AREP in which it receives APDUs produced by a publisher

3.2.32
sequence number
consecutive number used to determine whether a receiving message-data from other nodes has been previously received and is redundant

Note 1 to entry: The value is set to 0 (zero) at the time of the network-startup and is increased by 1 (one) every time on a message-data transmission completed.

3.2.33
server
<communication view> role of an AREP in which it returns a confirmed service response APDU to the client that initiated the request

3.2.34
server
<object view> object which provides services to another (client) object

3.2.35
token-frame
frame used for hand-over the token between Type 26 nodes

3.2.36
transaction code
code used to discriminate and identify each data transmission frame, and which represents the transaction of a receiving frame

3.2.37
trigger-frame
frame used in the network-startup state for a Type 26 node notifying of its entrance into a new participation-request over a Type 26 fieldbus network

3.2.38
valid-linking state
state in which a token is consecutively circulated among nodes over Type 26 fieldbus network

3.2.39
version of sequence number
version number of the sequence number in a receiving message-data from other nodes used to determine whether the receiving message-data is valid and is not overdue

Note 1 to entry: The value is set at the time when a Type 26 node completes to participate in a Type 26 fieldbus network, and is maintained until the node drops out of the Type 26 fieldbus network.

3.2.40
virtual-address-space
virtual memory space with pseudo logical address, used for the message-transmission

3.3 Abbreviations and symbols

3CWT	Waiting time period for 3 laps of token circulation
AE	Application Entity
AL	Application Layer
ALME	Application layer management entity
ALP	Application layer protocol
AP	Application Process
APDU	Application Protocol Data Unit
APO	Application Process Object
AR	Application Relationship
AREP	Application Relationship End Point
ASCII	American Standard Code for Information Interchange
ASE	Application Service Element
AWT	Waiting time period for receiving message-acknowledge
ASN.1	Abstract Syntax Notation One
BCD	Binary Coded Decimal
BCT	Broadcast transmission
BER	Basic Encoding Rule
BSIZE	Current-fragment-block-size
CBN	Current-fragment-block-number
C_AD1	Data-head-address on common-memory-area-1
C_AD2	Data-head-address on common-memory-area-2
CM	Common Memory
cnf	confirmation primitive
C_SZI	Data-size on common-memory-area-1
C_SZ2	Data-size on common-memory-area-2
DA	Destination-address
DCS	Distributed Control system
DL	Data-link
DLC	Data-link Connection
DLCEP	Data-link Connection End Point
DLL	Data-link layer
DLPDU	Data-link Protocol Data Unit
DLSAP	Data-link Service Access Point
DLSDU	Data-link Service Data Unit
DNA	Destination-node-number
DNS	Domain name service
FA	Factory Automation
FAL	Fieldbus Application Layer
FIFO	First In First Out
HMI	Human-machine interface
H_TYPE	Header-type
ID	Identifier

IEC	International Electrotechnical Commission
ind	indication primitive
IP	Internet protocol
ISO	International Organization for Standardization
LME	Layer management entity
LSB	least significant bit
MAC	Media Access Control
M_ADD	Message-offset-address in the virtual-address-space
MAJ_VER	FA-link protocol version: major-version
M_CTL	Message-control
MFT	Allowable-minimum-frame-interval-time
MIN_VER	FA-link protocol version: minor-version
MSB	most significant bit
MSN	Manufacturer's model name of a Type 26 node
M_SZ	Message-data-size in the virtual-address-space
M_RLT	Message result
NC	Numerical Controller
NDN	Node name of Type 26 node
OSI	Open Systems Interconnection
PAT	Participation-request-frame-acceptance-time
PC	Personal Computer
PDU	Protocol Data Unit
PhL	Ph-layer
PLC	Programmable Logic Controller
PPT	Peer-to-peer transmission
P_TYPE	Protocol-type
PWT	Participation-request-frame-transmission-waiting-time
RCT	Allowable-refresh-cycle-time
req	request primitive
RMT	Refresh-cycle-measurement-timer
rsp	response primitive
RC	Robot Controller
RT	Real Time
SA	Source-address
SAP	Service Access Point
SDU	Service Data Unit
SEQ	Sequence-number
SNA	Source-node-number
TBN	Total-fragment-block-number
TCD	Transaction code
TCP	Transmission Control Protocol
TDT	Joining-token-detection-time
TBN	Total-fragment-block-number
TFL	Total frame length in octet including both of header and data
THT	Token-holding-time
TrWT	Waiting time period before sending out a trigger-frame

TW	Token-watchdog
UDP	User Datagram protocol
ULS	Upper layer status
VDN	Vender code of Type 26 node
V_SEQ	Version of sequence number

3.4 Conventions

3.4.1 Conventions used in state machines

Protocol sequences are described by means of state machines.

The state machine description and the meaning of the elements of a state machine description are defined in tabular form as shown in Table 1.

Each row of the state table represents a state transition. The first column shows the state transition name or number. The second column shows the current state. The third column shows the events, conditions and actions. The fourth column shows the next state. When an event or condition is fulfilled, the action is performed and the state machine transitions to the next state.

Table 1 – Conventions used for state machines

#	Current state	Event / condition => action	Next state
Name or number of this state transition	The current state to which this state transition applies	Events or conditions that trigger this state transaction. => The actions that are taken when the above events or conditions are met. The actions are always indented below events or conditions	The next state after the actions in this transition is taken

The conventions used in the state machines are shown in Table 2.

Table 2 – Conventions used in state machine

Notation	Description
:=	Value of an item on the left is replaced by value of an item on the right. If an item on the right is a parameter, it comes from the primitive shown as an input event. xxx is a parameter name. Example: Identifier:= reason means value of a 'reason' parameter is assigned to a parameter called 'Identifier'. "xxx" Indicates fixed value Example: Identifier:= "abc" means value "abc" is assigned to a parameter named 'Identifier'.
=	A logical condition to indicate an item on the left is equal to an item on the right
<	A logical condition to indicate an item on the left is less than the item on the right
>	A logical condition to indicate an item on the left is greater than the item on the right
<>	A logical condition to indicate an item on the left is not equal to an item on the right
&&	Logical "AND"
||	Logical "OR"
!	Negation operator

Notation	Description
+, -, x, /	Arithmetic operator
;	Breakpoint
- – “comment” or /* “comment” */	Comments out conditions and actions. The “- – “comment” " type notation is used in one-line comments, and the “/* “comment text */ “ type notation is used in one or multiple-line comments.

Further the following conventions apply.

- The construct described below as an example allows the execution of a sequence of actions in a loop within one transition. The loop is executed for all values from start_value to end_value.

  Example:

  ```
  for (Identifier:= start_value to end_value)
      actions
  endfor
  ```

- The construct described below as an example allows the execution of alternative actions depending on some condition (which might be the value of some identifier or the outcome of a previous action) within one transition.

  Example:

  ```
  If (condition)
  then
      actions
  else
      actions
  endif
  ```

- Notations as defined in C language according to ISO/IEC 9899 is used to describe conditions and actions.

3.4.2 Convention for abstract syntax description

This document uses a subset of ASN.1 according to ISO/IEC 8824-1 for the description.

The description is provided by means of a combination of SEQUENCE type, CHOICE type, basic types and subtype elements defined in ASN.1.

3.4.3 Convention for reserved bits and octets

The term "reserved" is used to describe bits in octets or whole octets for indicating that certain values within the range of a parameter are reserved for future extensions. All the bits or the octets shall be set to zero at the sending side and is not validated intentionally at the receiving side except it is explicitly stated that a state machine shall validate the bits or octets reserved.

3.4.4 Conventions for bit description in octets

Each bit in an octet is identified by a number and is described as Bit n. Figure 1 shows the bit identification in an octet.

When specifying bit identification in multiple octets, the LSB of the lowest octet is identified as Bit 0, and the bit identification numbers are assigned in an ascending order. Figure 2 shows the bit identification in four octets as an example.

Bits are grouped as group of bits. Each bit or group of bits shall be addressed by its Bit Identification. When specifying multiple bits sequentially located, the range separator ".." is used. A group of bits of Bit 0 through Bit 4 denotes Bit 0..4.

Alias names are used for each bit or group of bits. The grouping of individual bits is in ascending order without gaps. The values for a group of bits are provided as binary, decimal or hexadecimal values, and the value is only valid for the group bits.

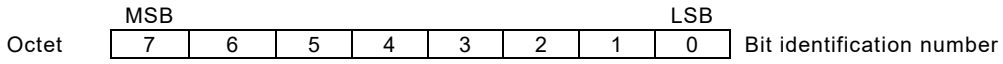


Figure 1 – Bit identification in an octet

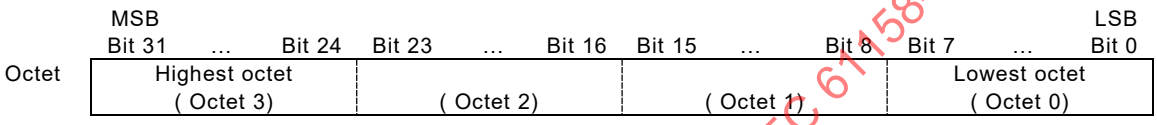


Figure 2 – Bit identification in multiple octets (four-octet case)

4 FAL syntax description

4.1 General

This document specifies the Application layer protocol of Type 26 fieldbus that is essential for the ISO/IEC/IEEE 8802-3 based FA Link network (FL-net[1]) which will be incorporated as one of the communication networks for the Real-time Ethernet (RTE) defined in IEC 61784-2.

The Type 26 fieldbus meets the industrial automation market objective of providing predictable time-deterministic and reliable time-critical data transfer and means, which allow co-existence with non-time-critical data transfer over the ISO/IEC/IEEE 8802-3 series communications medium, for support of cooperation and synchronization between automation processes on field devices in a real-time application system.

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.

Plant production may be compromised due to errors, which could be introduced to the control system if the network does not provide a time-critical response. Therefore the following characteristics are required for a time-critical control network:

– A deterministic response time between the control device nodes.
– Ability to share process data seamlessly across the control system.

The Type 26 fieldbus is applicable to such an industrial automation environment, in which time-critical communications are primarily required.

[1] FL-net is the trade name of JEMA/FL-net: The Japan Electrical Manufacturers' Association / the Factory Automation Link network. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance does not require use of the trade name. Use of the trade name requires permission of the trade name holder.

4.2 Overview of Type 26 fieldbus

4.2.1 Application field and Common-memory

The Type 26 fieldbus is applicable to a controller level network for efficiently executing the control and state data exchange between various kinds of field devices such as Programmable Logic Controllers (PLC), Robot Controllers (RC), Numerical Controllers (NC), Factory Automation (FA) controllers, Distributed Control systems (DCS), Personal Computers (PC) for the control purpose and the human-machine interfaces and so forth, in industrial automation control systems in which a time-criticality for their data exchanges is required.

The communications between these field devices also require simplicity in actual application programming as well as the time-critical transfer of the control and state data required among these field devices. To meet such requirement the Type 26 fieldbus provides so-called Common-memory (CM) which provides a function to share the control and state data seamlessly across an industrial control network system.

The CM is a virtual common memory accessible with logically unique address across the Type 26 nodes in a Type 26 fieldbus network, and the APs running over each Type 26 node can handle the CM as a global common memory shared among the Type 26 nodes over the Type 26 fieldbus network.

Cyclic-data transmission is used for refreshing and equalizing the contents of the CM among the Type 26 nodes by means of a cyclic broadcasting performed at a constant rate after a Type 26 node writes a data peace on a pre-allocated memory area of the CM for the Type 26 node.

The data distribution/refreshing function to all Type 26 nodes provided by the CM by means of the cyclic-data transmission enables each node to share the same data with each other, and to make good use of the data; i.e. a memory space allocated to each node is a container for application data in general use and provides flexibility to apply in a variety of industrial application processes.

In operational view of the CM, the memory area in the CM allocated to a certain node is used as the transmission area of the node by a cyclic-data transmission as a sender. On the other hand as receiving nodes of the cyclic-data transmission, the memory area of the sender node is used as a receiving area for the sender nodes.

Figure 3 shows the data sharing with the CM.

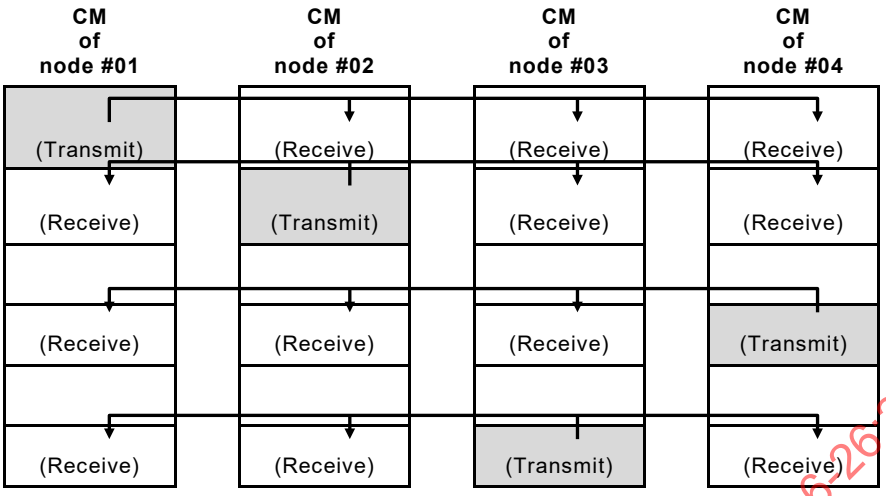


Figure 3 – Data sharing with the CM

4.2.2 Structure of Type 26 protocol

The application layer exploits the services of the data-link layer immediately below, in terms of the "three-layer" Fieldbus Reference Model which is based in part on the OSI Basic Reference Model. The OSI reference model provides a layered approach to communications standards, whereby the layers can be developed and modified independently. IEC 61158 specifies functionality from top to bottom of a full OSI stack and, potentially, some functions for the users of the stack.

Figure 4 shows the protocol stack for a Type 26 fieldbus, the OSI layers and the equivalent layers in the IEC 61158 basic fieldbus reference model.

Functions of the intermediate OSI layers, layers 3 through 6, are consolidated into the IEC 61158 data-link layer for a Type 26 fieldbus as the lower layer of the FA Link protocol. The data-link layer is that of a general-purpose Ethernet with UDP (see RFC 768), TCP (see RFC 793), IP (see RFC 791) and Ethernet layer (see RFC 894, ISO/IEC/IEEE 8802-3) as the main components. Type 26 fieldbus utilizes that of the services provided with a general-purpose Ethernet protocol suite.

The application layer for Type 26 fieldbus is the main component of the FA Link protocol stack, and is specified as the upper layer of the FA Link protocol. The application layer is mapped directly on the data-link layer as the lower layer of the FA Link protocol.

Two kinds of data transmissions are defined; i.e. Cyclic-data transmission and Message-data transmission. The cyclic-data and the message-data transmissions are the data transmission of UDP connectionless based. Data transmission based on connection-oriented TCP is only for the General-purpose-command-server transmission.

This document defines the protocol specification, and the service elements and services are defined in IEC 61158-5-26.

<table>
<tr><th colspan="2">OSI layer</th><th></th><th>IEC 61158 layer</th></tr>
<tr><td>7</td><td>Application</td><td></td><td rowspan="3">Application

IEC 61158-5-26
IEC 61158-6-26</td></tr>
<tr><td>6</td><td>Presentation</td><td>null</td></tr>
<tr><td>5</td><td>Session</td><td>null</td></tr>
<tr><td>4</td><td>Transport</td><td></td><td>Data-link

RFC 768, (RFC 793)</td></tr>
<tr><td>3</td><td>Network</td><td></td><td>RFC 791
RFC 919, RFC 922
(RFC 792, RFC 950)</td></tr>
<tr><td>2</td><td>Data-link</td><td></td><td>RFC 894/
ISO/IEC/IEEE 8802-3 MAC
(RFC 826)</td></tr>
<tr><td>1</td><td>Physical</td><td></td><td>Physical

ISO/IEC/IEEE 8802-3 Phy</td></tr>
</table>

Figure 4 – Protocol stack for Type 26 fieldbus

4.2.3 Structure of Type 26 FAL

The FAL is defined as a set of object-oriented ASEs. Each ASE specification is composed of three parts: the class definitions, the services and the protocol specification. The first two are contained and defined in IEC 61158-5-26.

The FAL ASEs and their architectural relationships for Type 26 FAL are shown in Figure 5.

Each ASE provides the services as follows:

a) The cyclic-data ASE provides the cyclic data communication service with Type 26 specific common-memory defined in 6.4 of IEC 61158-5-26, and the cyclic-data transmission is performed at a constant rate for updating the common-memory;

b) Message data ASE provides the message-data communication service between nodes and the message-data transmission is performed on demand by the ALS-user in order to intercommunicate between the APs running on remote nodes;

c) The load measurement ASE provides services to conduct the start and the termination of the communication load measurement on the target node, and provides the measurement result containing the token holding time related and the general purpose communication load related statistical information on that node available for analyzing the total communication load condition over the Type 26 fieldbus network;

d) The General purpose command server ASE provides the general purpose command server communication service, and provides the facilities for setting and reading the values of various network visible APOs of other nodes, and for requesting the start and the stop of the communication load measurement to other nodes;

e) The Network management ASE provides services to access Type 26 APO attributes; modify the APO instances; configure the nodes; inform the status report on the unexpected events, the errors and the status changes of the nodes; manage network visible APOs accessed through the FAL.

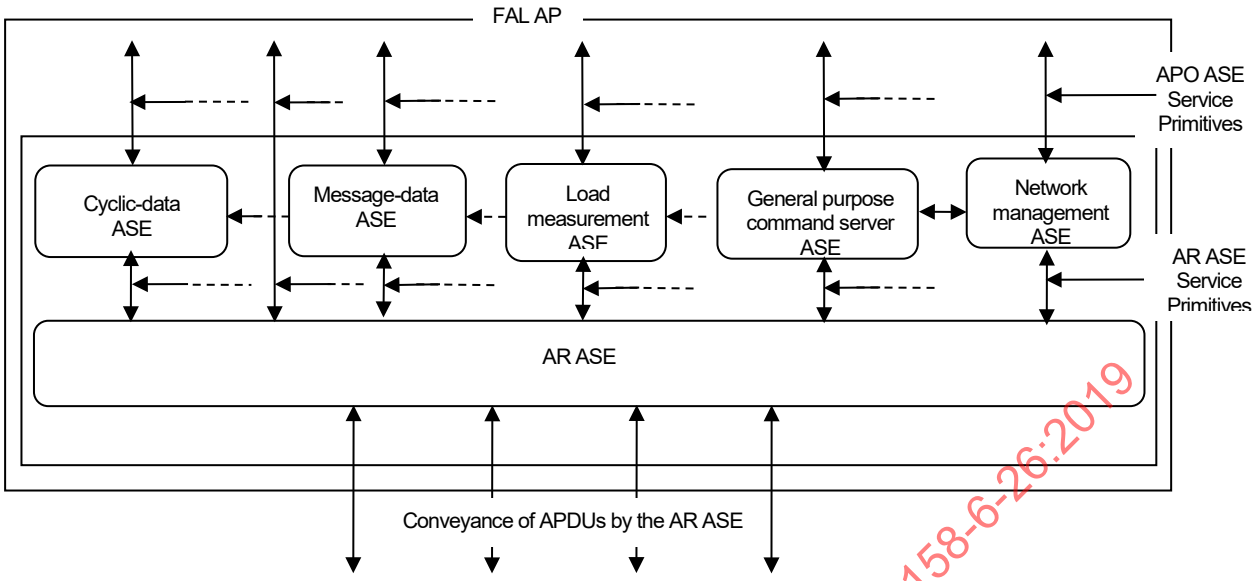


Figure 5 – The structure of ASEs for Type 26 FAL

4.2.4 Data link layer

Two types of addressing exist for DL-addressing in terms of the Fieldbus three layer reference model, one is over UDP and other is over TCP protocol channels which are represented as the T- profile in terms of the OSI reference model.

The cyclic-data and the message-data transmissions are the data transmission of UDP connectionless based. The message-data transmission based on connection-oriented TCP is only for the General-purpose-command-server transmission.

The cyclic-data and the message-data transmissions utilize the local broadcast medium/mechanism over the UDP and the TCP protocols, and are restricted generally to the local link. Requirement for the Data link layer as the FAL lower layer shall specify the broadcast communication function as the mandatory fundamental function.

With respect to the Ethernet MAC address, the default set to the Ethernet MAC interface by the device vendor is used as the MAC address, and for the IP address, class C of IP address defined in RFC 796 is assigned and used. The default value of the IP address is specified as 192.168.250.N with the value of 255.255.255.0 as the subnet mask, and the value of N shall be identical to the Node number of 1 to 254.

4.3 Operating principle

4.3.1 Overview

A token controls the right to send the data out of a Type 26 node to a Type 26 fieldbus network. The node which holds (possesses) the token can primarily perform the data transmission over the Type 26 fieldbus network.

The token is passed by Type 26 nodes, and as the token is passed from node to node a logical ring is formed. Logical ring maintenance functions within the Type 26 nodes are provided for ring initialization, lost token recovery, multiple tokens, new node addition, node drop-out, entering and exiting ring, monitoring and detection fault and so forth.

Operation in normal, steady state on the node consists of a data transmission phase and a token transmission phase. The two kinds of data transmissions for Type 26 nodes are the cyclic-data transmission and the message-data transmission.

The cyclic-data transmission is primarily carried out by the node every time the node obtains the token; on the other hand the message-data transmission is dependent on request and the communication load condition at that time. After the node has completed transmitting any data, the node passes the token to the successor by sending out a token frame.

4.3.2 Logical ring maintenance

4.3.2.1 Token passing

A token is passed by the nodes, and as the token is passed from node to node a logical ring is formed. Figure 6 shows a token circulation on a logical ring.

The token transfer is performed by sending a token frame out of a node (predecessor) to its successor after the predecessor has completed sending any data frames. The token frame contains a destination node number (DNA) and a source node number (SNA). The DNA indicates the node number of successor, and the SNA indicates the node number of predecessor.

Type 26 fieldbus is in general a broadcast type, and every Type 26 node receives the token frame. The node has the right to send out the data by recognizing and accepting the token frame of which the DNA number is identical with the node number.

The token passing is always sequential in a logical sense. During normal, steady-state operation, a data transmission phase and a token transmission phase are performed. After the node has completed transmitting any data, the node passes the token to the successor by sending out a token frame.

The token or the right to send out data passes from node to node in a logically circular fashion. The token or the right to send out data is circulated from node to node in ascending numerical order of the node number.

The node which holds the token and has the largest node number in a Type 26 fieldbus network shall pass the token to the node which has the smallest node number. Figure 6 shows a Type 26 fieldbus network which is consisting of Node #1 to Node #N. Each node number of the Node #1 to the Node #N is 1 to N respectively in ascending numerical order. In that configuration, the Node #N which holds the token and has the largest node number shall pass the token to the Node #1 which has the smallest node number in order to maintain the logical ring of the Type 26 fieldbus network.

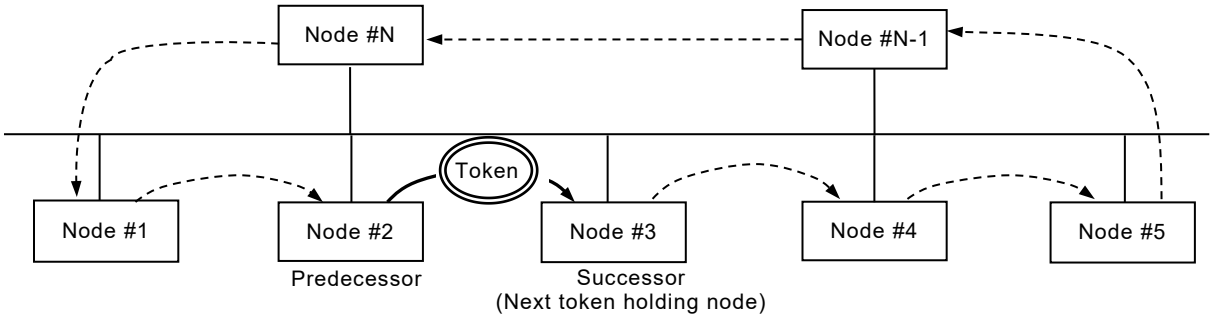


Figure 6 – A token circulation on a logical ring

4.3.2.2 Right to transmit

A node can hold the right to transmit data over the Type 26 fieldbus network after obtaining the token until the time that the node shall hand over the token to the next node or the successor.

Operation in normal, steady state on the node consists of a data transmission phase and a token transmission phase. In the data transmission phase, two kinds of data transmissions exist; i.e. the cyclic-data transmission and the message-data transmission. The cyclic-data transmission is primarily carried out by the node every time the node obtains the token; on the other hand the message-data transmission is dependent on request and the communication load condition at that time.

During the above process, the data transmission is performed as follows:

a) The token-holding-time (THT) controls and limits the time period within which every node is able to be holding the token. The time period of holding the token by the nodes shall not exceed the time period of the THT;

b) While holding the token, the node can send out one or more cyclic-data frames every time and further can send out at most one message-data frame;

c) After the node has completed transmitting any data within the time period of the THT, the node shall pass the token to the successor by sending out a token frame;

d) The token frame is never sent out upon failure that the time period of holding the token exceeds the time period of the THT.

NOTE Nodes which do not holde the token are exceptionally able to send out Type 26 specific control frames. A new node can send out these frames during the entering state in order to join in an operating Type 26 fieldbus network or the logical ring in operation. In addition, a node can send out a new token frame in order to reform and recover the logical ring, after it has been disrupted by one of the following events: token lost, node dropped out of the logical ring, failure happening within nodes, and so forth.

4.3.2.3 Logical ring recovery

When a node recognizes and determines the logical ring is disrupted whether or not it holds the token, sends out a token frame in order to reform and recover the logical ring.

Each node is monitoring the state of a token circulation on the logical ring by means of two observational timers that are the refresh-cycle-measurement-timer (RMT) and the token-watchdog-timer (TW).

The RMT is provided with each node and is an observational timer to measure the refresh-cycle-time which is the elapsed time period from the time after obtaining the token by a node until the time the node obtains the token once again. When the RMT overruns the limit of allowable-refresh-cycle-time (RCT) set to each node and common to all nodes, the RMT is considered to be time out.

The TW-timer is provided with each node and is used as an observational timer to measure the elapsed time period from the time a node obtains the token until the time the node sends out a token frame over the fieldbus network.

Each node has the maximum observational time limit to the TW-timer. Each node has the knowledge on the TW values of other nodes, for instance by means of checking the APDU header received from other nodes, and the node can utilize these TW values of other nodes to detect some kind of failure by means of checking the elapsed time period during which a node holds the token.

When the TW-sum-time in the case of using the maximum observational time limit to the TW-timer of each node overruns the limit of the sum total of the maximum observational time limit to the TW-timer of each node which resides sequentially on the logical ring from a token

holding node to this node in ascending numerical order of the node number, the TW-timer of some node is considered to be time out.

A token frame shall be sent out of a node upon time out both of the RMT and the TW-timer at a time. Both timers out at the same time results from the logical ring disrupted, so that the token frame shall be sent out to reform and recover the logical ring.

Figure 7 shows the sequential process for the logical ring recovery in the case that the node #3 dropped out from the logical ring of five nodes. The sequential process is as follows:

a) The node #1 passes the token to the node #2,

b) The node #2 passes the token to the node #3,

c) Since the node #3 has not responded, a token frame shall be sent out from the node #4 to the node #5 after the TW-timer and the RMT out of the node #4,

d) The node #5 passes the token to the node #1,

e) The node #1 passes the token to the node #2,

f) Since the node #3 has dropped out of the logical ring, the node #2 shall pass the token to the node #4,

g) The token is passed to the node #5 from the node #4. The token circulates on new logical ring of node #1, #2, #4 and #5.

NOTE When the nodes receive a token frame from the node which is identified not as a member node in the logical ring and then is identified as a new node, the TW value for the node is set to the maximum time value of 255 ms.

When the nodes receive a token frame from the node which is once a member node of the logical ring, the TW value for the node is set to the same value which is used and set to as the TW value of the node at that time.

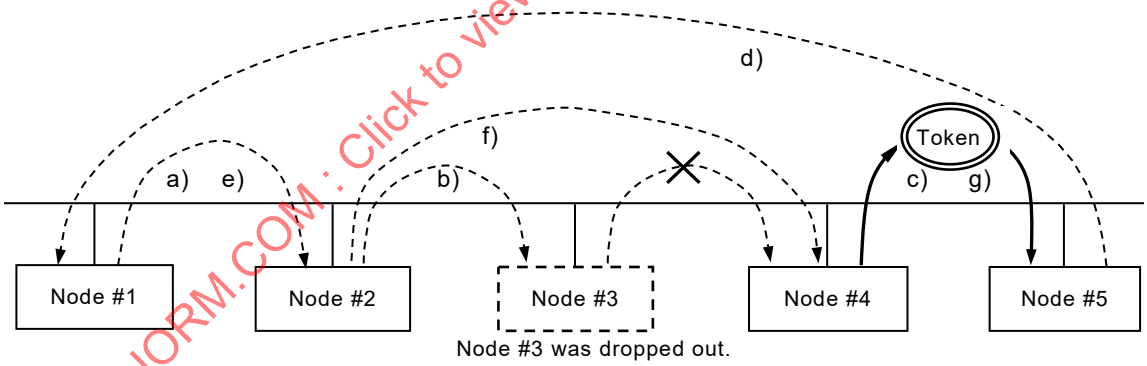


Figure 7 – Logical ring recovery

4.3.2.4 Multiple tokens

There are two types of conditions for a node to handle the problem of multiple tokens:

a) When a node is holding the token and receives a token frame of which the DNA number is not identical to the node number and is smaller than that of the node, the node shall abandon the right to send out the data; shall give priority to the node with the DNA number and shall enter into the waiting for the token state;

b) On the other hand, when a node with the token receives a token frame with the DNA number which addresses to other node and is not smaller than that of the node, the node shall discard the token frame received and shall maintain the state in which the node is holding the token and is holding the right to send out the data.

4.3.3 Node addition

4.3.3.1 Overview

A new node in the entering state into a logical ring shall monitor the activity on the Type 26 fieldbus network until the joining-token-detection-time timer (TDT-timer) is timed out. The default observational time period is 3 000 ms and the value is set to the TDT-timer.

There are two kinds of conditions to start the process for node addition; one is under the condition in the network start-up and the other is under the condition in the in-ring start-up state:

a) During the time period when the node does not detect any token frames, the node recognizes and determines that the logical ring over the Type 26 fieldbus network is not yet formed and is in the network start-up state;

b) On the other hand, when the node detects any token frames, the node determines that the logical ring over the network has already formed and is in operation, and then the node shall enter into the in-ring start-up state.

4.3.3.2 Node addition in the network start-up state

This process and the procedures are identical to form a new logical ring or to start up a new Type 26 fieldbus network.

There are two types of conditions for a new node to enter into this process; i.e. another node is simultaneously entering and the new node alone enters into this process under the condition in the network start-up state.

Figure 8 shows an example of the sequential timing diagram in the case that a node enters into this process simultaneously with another node under the condition in the network start-up state. Figure 9 shows the sequential timing diagram in the case that a node is alone and enters into this process under the condition in the network start-up state.

The process and the procedures are as follows:

a) A new node shall start the Trigger-frame-transmission-waiting-time timer (TrWT-timer) after the TDT-timer out without any token frames received. In case of no reception of any trigger frames from another node until the TrWT-timer out, a trigger frame shall be sent out of the new node as shown in Figure 9;

b) On the contrary, the new node has received any trigger frames from another node during the time period of the TrWT. In this case the trigger frame shall not be sent out of the new node, and the new node shall start both of the Participation-request-frame-transmission-waiting-time timer (PWT-timer) and the participation-request-frame-acceptance-time timer (PAT-timer) upon reception of a trigger frame from another node as shown in Figure 8;

c) After the TrWT-timer out, the new node shall start both of the Participation-request-frame-transmission-waiting-time timer (PWT-timer) and the participation-request-frame-acceptance-time timer (PAT-timer) in case of no reception of any trigger frames during the time period of the TrWT as shown in Figure 9;

d) Each node has different values of the time period of the TrWT and the PWT, and the values are set as follows:

   – TrWT = R x 4 (ms):

     R is the remainder of the node number divided by 8;

   – PWT = (the node number) x 4 (ms).

   The new node shall send out a participation-request frame upon the PWT-timer out;

e) The new node shall wait for reception of any participation-request frames from another node until the PAT-timer out. The default observational time period is 1 200 ms and the value is set to the PAT-timer;

f) During the PAT time period upon reception of any participation-request frames, each node shall check and validate the contents of receiving participation-request frames; i.e. shall check the overlap of each common-memory-area claiming for its own use by each node and the duplication of the node number of other nodes; and further shall enrol the sender node related information with the validated common-memory area information and so on into the participating-node-management information table. The participating-node-management-information is updated upon reception of any participation-request frames;

g) Upon the PWT-timer out, the new node shall send out the participation-request frame;

h) Upon the PAT-timer out, according to the contents of the participating-node-management-information table new logical ring shall be formed; and then the node with the minimum node number shall send out a token frame first as shown in Figure 8;

i) On the contrary, in the case of no reception from other nodes until the PAT-timer out, the TDT-timer shall be started again and the above mentioned process and procedures shall be followed and repeated until any trigger frames, any participation-request frames or any token frames from other nodes is received as shown in Figure 9. After repeating three cycle times or sending out the participation-request frame three times, upon the PAT-timer out, the node shall recognize and determine that only the node exists and attempts alone to form a new logical ring; and then shall notify the FAL-user of the node status with a flag of waiting for any receptions from other nodes;

j) The node which detects and identifies the overlap of the node number with other nodes by means of validating the participation-request frame received from another node shall not enter and participate in the new logical ring; and shall not send out any data and shall discard any receiving frames. Error information with the duplicated node number shall be notified to the FAL-user;

k) The node which detects and identifies the overlap of the common-memory area with other nodes by means of validating the participation-request frame received from another node shall enter and participate in the new logical ring; shall set the common-memory area unavailable: i.e. set to zero (0) the head-address and the size of the common memory area for own use of the node; and shall not send out any cyclic-data. Error information with the overlap of the common-memory area shall be notified to the FAL-user; the validity of the contents of the common-memory shall be flagged as invalid and unavailable.

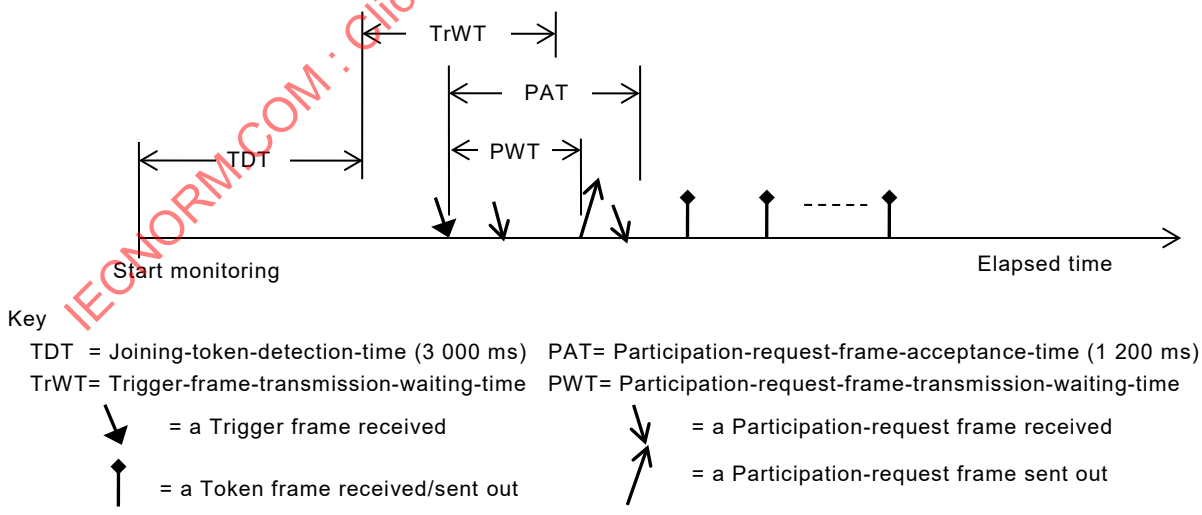
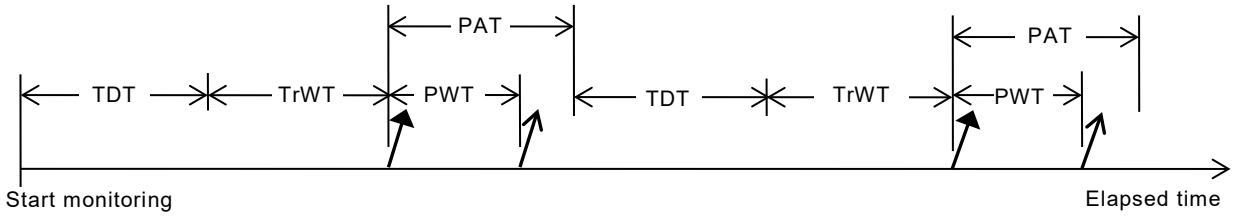


Figure 8 – An example in case of start simultaneously with another node



Key

TDT = Joining-token-detection-time (3 000 ms) PAT= Participation-request-frame-acceptance-time (1 200 ms)

TrWT = Trigger-frame-transmission-waiting-time PWT= Participation-request-frame-transmission-waiting-time

= a Trigger frame sent out

= a Participation-request frame sent out

Figure 9 – Start alone case

4.3.3.3 New node addition in the in-ring star-up state

A new node in the entering state into a logical ring shall monitor the activity on the Type 26 fieldbus network until the TDT-timer out. During the TDT time period when the node detects any token frames, the node determines that the logical ring over the network has already formed and is in operation, and then the node enters into the in-ring start-up state.

Figure 10 shows an example of the sequential timing diagram for a new node of node #5 in the in-ring start-up state into 4-node logical ring in operation.

The process and the procedures are as follows:

a) The node shall keep the monitoring until the token circulates on the logical ring three times after receiving the token frame with the minimum node number in the logical ring;

b) During the three-lap time period of the token circulation on the logical ring, the node shall check and detect the overlap of the common-memory-area claiming for its own use with the area used by other nodes and the duplication of the node number to other nodes by means of the validation of any receiving frames containing these information; and after the validation of any receiving frames, the node shall enrol the sender node related information with the validated common-memory area information and so on into the participating-node-management information table. The participating-node-management-information is updated during this time period;

c) After the three-lap time period of the token circulation (3CWT time period) with no duplicated node number, the node shall start the PWT-timer to send out a participation-request frame to enter into the logical ring;

d) Upon the PWT-timer out, the node shall send out a participation-request frame even though the node is not holding the token;

e) The node which detects and identifies the duplication of the node number to other nodes shall not send out any participation-request frames; and shall not enter into the logical ring. The node shall notify the FAL-user of the error information with the duplicated node number and the node-status with a flag of node number duplication;

f) The node which detects and identifies the overlap of the common-memory area with other nodes shall enter and participate in the new logical ring; shall set the common-memory area unavailable: i.e. set to zero (0) the head-address and the size of the common memory area for own use of the node; and shall not send out any cyclic-data. Error information with the overlap of the common-memory area shall be notified to the FAL-user; and the validity of the contents of the common-memory shall be flagged as invalid and unavailable;

g) After sending out a participation-request frame, in case of no reception of any token frames addressed to this node until the 3CWT-timer out, the node shall attempt and follow again the above mentioned process and the procedures to enter into the logical ring;

h) After sending out a participation-request frame, in case of no reception of any token frames until the 3CWT-timer out, the node shall attempt and follow again the above mentioned process and the procedures to enter into the logical ring;

i) When a node on a logical ring receives any participation-request frames, the node shall update and enrol the receiving node related information into the participating-node-management-information after validation without the overlapping of the common-memory address space and further the duplication of the node number. Detected error information shall be notified to the FAL-user.

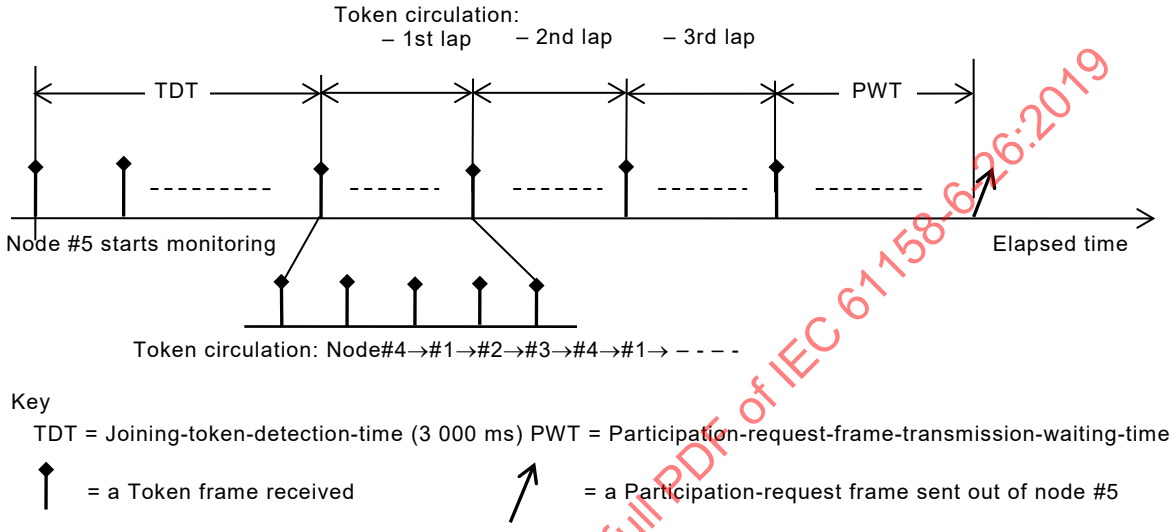


Figure 10 – Node addition: in-ring start-up state

4.3.4 Node in a logical ring

When a node on a logical ring in normal, steady state receives any participation-request frames and/or any final cyclic-data frames which contain the fragmented cyclic-data and are sent out last by the nodes, the node shall update and enrol the receiving node related information into the participating-node-management-information even though the sender node number is not identified after validation without the overlapping of the common-memory address space and further the duplication of the node number. Detected error information shall be notified to the FAL-user.

4.3.5 Node drop-out

Node drop-out or node exiting out of the logical ring is as follows:

a) Each node on a logical ring in normal, steady state shall check the sender node number of any token frames every time on the reception; and shall detect whether no reception of the token frame from a certain node continues for three consecutive token circulation periods; and shall recognize and determine the corresponding certain node is removed and dropped out of the logical ring;

b) When the node determines the corresponding certain node is dropped out of the logical ring, the node shall update the corresponding certain node related information within the participating-node-management information to change the status as the corresponding certain node removed.

c) When the node recognizes and determines all other nodes are removed during the token circulation periods and only the node itself exists, the node shall move into the initial state and take the procedures specified in 4.3.3.2 and 4.3.3.3.

d) Each node on a logical ring in normal, steady state shall monitor and check any token frames addressed to the node. In the case that no reception of the token frame addressed

to the node from other nodes continues for three consecutive token circulation periods, the node shall recognize and determine the node is dropped out of the logical ring. The node shall move into the initial state and take the procedures specified in 4.3.3.2 and 4.3.3.3.

4.3.6 Data transmission

4.3.6.1 Overview

Operation in normal, steady state on the node consists of a data transmission phase and a token transmission phase.

In the data transmission phase, two kinds of data transmissions are performed:

– Cyclic-data transmission;
– Message-data transmission.

4.3.6.2 Cyclic-data transmission

4.3.6.2.1 Overview

The Type 26 fieldbus is applicable to a controller level network for efficiently executing the control and state data exchange between various kinds of field devices in industrial automation control systems.

The communications between field devices requires simplicity in actual application programming as well as the time-critical transfer of the control and state data among field devices. To meet such a requirement, the CM provides a function to share the control and state data seamlessly across an industrial control network system.

The CM is a virtual common memory accessible with logically unique address across the Type 26 nodes, and the APs running over each node can handle the CM as a global common memory shared among the nodes over the Type 26 fieldbus network.

Cyclic-data transmission is used for refreshing and equalizing the contents of the CM among the nodes by means of a cyclic broadcasting performed at a constant rate after a node writes a data peace on a pre-allocated memory area of the CM for each node.

The data distribution/refreshing function to all nodes in temporal and spatial coherency provided with the CM by means of the cyclic-data transmission performed at a constant rate enables each node to share the same data with each other and to make good use of the data; i.e. a memory space allocated to each node is a container for application data in general use and provides flexibility to apply in a variety of industrial application processes.

Figure 11 shows the data sharing with the CM.

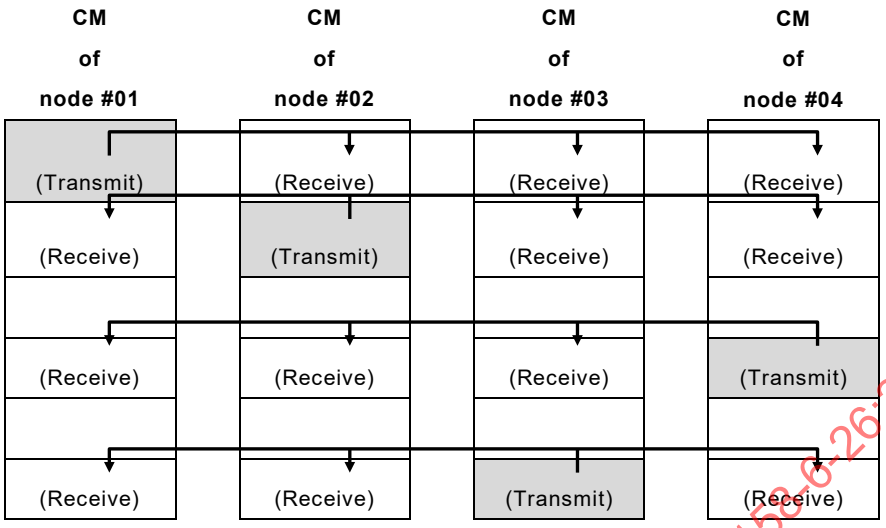


Figure 11 – Data sharing with the CM

4.3.6.2.2 Configuration of Common-memory

Figure 12 shows the configuration of the CM which is composed of two memory areas; i.e. the memory-area-1 and -2 with the memory size of 512 words and 8 192 words respectively. The total memory size of the CM is up to 8 704 words.

The memory area allocated to each node as the output/write area on one or both of the memory-area-1 and -2 is dependent on the implementation. Any memory size in word boundary is allocable from 0 (zero) to the maximum space size of the memory-area-1 and/or -2 to each node, and the memory space allocation is designated with the memory size and the head-address. The memory space allocated to each node shall not overlap with each other. And the allocated memory space is identical to the cyclic-data transmission area by the node.

In case of total zero memory space allocated to a node, it means no cyclic-data transmission to be carried out by the node. It is applicable to a mixture system of the nodes with and without the CM.

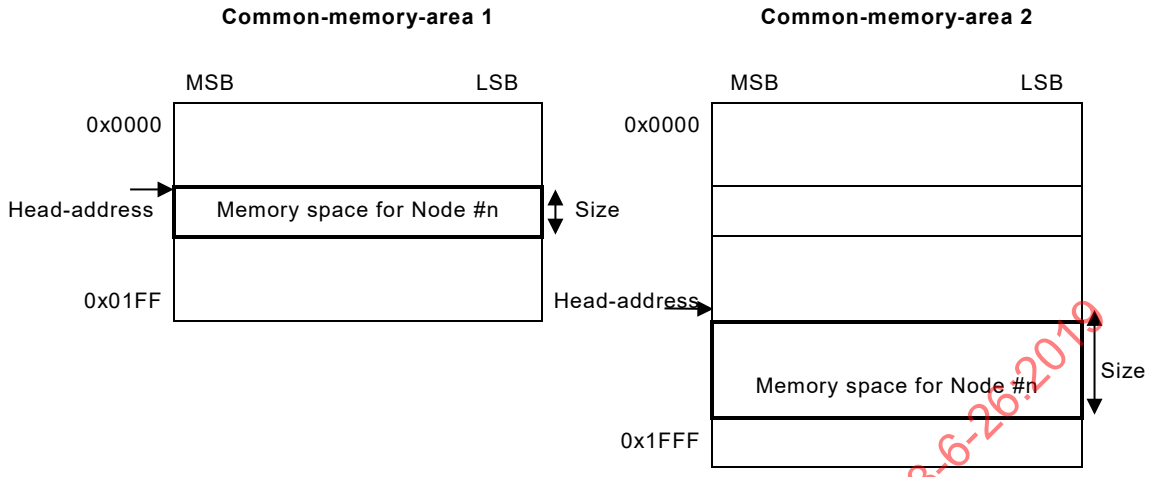


Figure 12 – Configuration of the Common-memory

4.3.6.2.3 Cyclic-data transfer

Cyclic-data is transferred by means of the cyclic-data transmission.

The memory area on the CM allocated to a certain node is used as the transmission area of the node by the cyclic-data transmission as a sender. On the other hand as receiving nodes of the cyclic-data transmission, the memory area of the sender node is used as a receiving area for the data from the sender node.

Cyclic-data transmission shall be performed each time the node obtains the token, and all the data on the CM area allocated to the node shall be sent out of the node.

Dependent on the total amount of data to be transferred, one or more cyclic-data frames are sent out each time the node obtains the token. In the case that the total amount of data is less than or equal to 1 024 octets, data resided in the memory space allocated for the node on the CM-area-1 and -2 shall be packed into a single cyclic-data frame and shall be sent out at a time. On the other hand in the case that the total amount of data is over 1 024 octets, the data shall be fragmented into multiple packets and shall be sent out by means of consecutive multiple cyclic-data frames while holding the token.

The total amount of data over 1 024 octets which resides in the memory space riding both on the CM-area-1 and -2 shall be fragmented into two or more packets for multiple cyclic-data frames to be sent out. The first cyclic-data frame shall contain all the data resided in the memory space on the CM-area-1 and all or part of the data resided in the memory space on the CM-area-2 but the total volume of data contained in the cyclic-data frame shall not exceed 1 024 octets. The following cyclic-data frames shall be sent out with the data by 1 024 octets until all the data is packed into multiple cyclic-data frames. The last cyclic-data frame, so-called “final-cyclic-data frame” shall contain the data of 1 024 octets or less size.

Figure 13 shows an example of the APDUs for multiple cyclic-data frames containing fragmented data. In this case since the total amount of data is over 1 024 octets, the data are fragmented and packed into three cyclic-data frames.

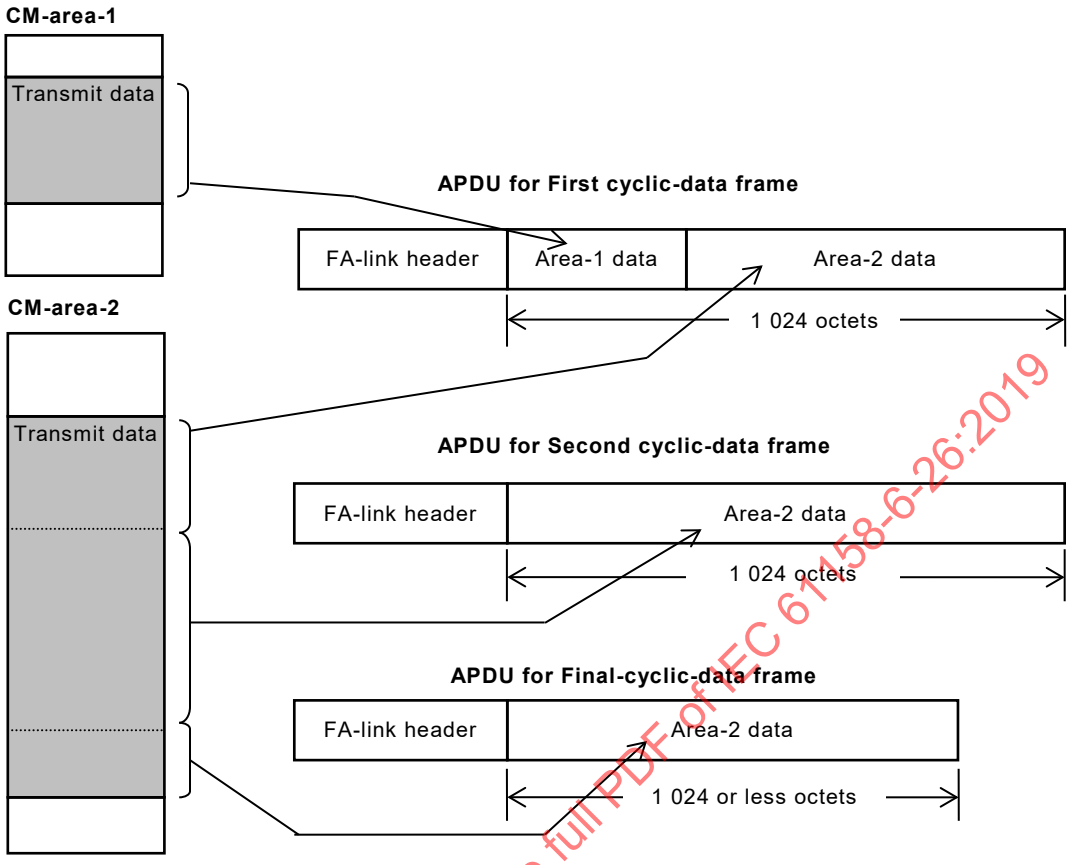


Figure 13 – APDUs of cyclic-data frames containing fragmented data

Each cyclic-data frame shall contain the information in the FA-link header to disassemble/assemble the cyclic-data APDU as follows:

- TBN: total-fragment-block-number;
- CBN: current-fragment-block-number;
- BSIZE: current-fragment-block-size;
- C_AD1: data-head-address on common-memory-area-1;
- C_AD2: data-head-address on common-memory-area-2;
- C_SZ1: data-size on common-memory-area-1;
- C_SZ2: data-size on common-memory-area-2.

Each node receiving a cyclic-data frame shall check the value of TBN in the FA-link header and determine whether the cyclic-data frame is fragmented or not. In the case of the cyclic-data frame fragmented, the receiving node shall store each received block into the receive buffer one by one upon reception of the fragmented cyclic-data frames; and after receiving the final-cyclic-data frame without error, all data received from other nodes shall be written into the corresponding memory address space on the common-memory-area-1 and -2.

In the case that each node fails to receive a series of the cyclic-data frames from other nodes without error, the node shall discard all the received data stored in the receive buffer without writing into the corresponding memory address space on the common-memory-area-1 and -2.

4.3.6.3 Message-data transmission

4.3.6.3.1 Overview

Message-data transmission is used by the message-data transfer service which is based on aperiodic data transfer and is sporadically occurred upon FAL-user request to transfer one or more messages to other nodes. Message-data transfer is performed by means of message-data transmission on the communication channel provided by the DL service of UDP (see RFC 768) or TCP (see RFC 793) as the lower layer of Type 26 FAL.

NOTE Message-data transmission in general is performed by means of connectionless UDP communication channels. The message-data transmission based on connection-oriented TCP channels is only for the General-purpose-command-server-communication which is utilized for intercommunication between the nodes and the general-purpose Human-Machine-Interface tool.

4.3.6.3.2 Message-data transfer

Message-data is transferred by means of the message-data transmission.

Message -data transmission shall be performed as follows:

a) When a node obtains the token and the message-data transfer is requested by the FAL-user, the node shall send out at a maximum one message-data frame before sending out the cyclic-data frames;

b) The maximum size of the message-data contained in one message-data frame shall be restricted to 1 024 octets excluding the FA-link header;

c) Two types of the message-data transmission are applicable: i.e. one-to-one transfer type and one-to-N broadcasting type. The destination node number (DNA) contained in the message-data frame of one-to-one type message-data transmission shall be specified from 1 to 254 as the receiver node number, on the other hand the DNA shall be set to 255 for one-to-N broadcasting type message-data transmission; and the node receiving the message-data frame shall check the received DNA and determines whether the message-data frame is one-to-one type or one-to-N type message-data transmission.

The one-to-one type message-data transmission is based on confirmed type data transmission; on the other hand the one-to-N type message-data transmission is of unconfirmed type.

The one-to-one type message-data transmission provides the functions for automatic retransmission and sequence number management. The automatic retransmission is performed, for instance upon no acknowledgement from the receiver node after sending out a message-data frame addressed to a receiver node. The sequence number management is utilized to check and determine whether a receiving message-data frame has been previously received and is redundant.

Table 3 summarizes the functions available to each type of message-data transmission.

Delivery confirmation and automatic retransmission is performed explicitly by means of the message-data transmission on connectionless UDP channels. On the other hand, delivery confirmation and automatic retransmission by means of the message-data transmission on connection-oriented TCP channels is performed implicitly. The TCP protocol provides these functions.

Table 3 – Available functions to message-data transfer on UDP channel

Message-data transmission	Delivery confirmation	Automatic retransmission	Sequence number management
One-to-one type	Available	Available	Available
One-to-N type	–	–	Available

4.3.6.3.3 Traffic control of message-data transmission

The traffic control function is performed primarily for the message-data transfer on connectionless UDP channels.

When a node obtains the token, the node shall send out at a maximum one message-data frame before sending out the cyclic-data frames and it shall be performed on the basis of the condition of communication load at that time. The condition of the communication load for the node possible to send out one message-data frame is as follows:

- The value of the RMT shall be less than or equal to 90 % value of the RCT, or
- The value of the RMT is greater than 90 % value of the RCT and is less than the value of the RCT and no message-data frame was sent out last time when the node obtained the token.

When the value of the RMT is equal to or greater than the value of the RCT, no message-data frame shall be sent out of the node.

4.3.6.3.4 Delivery confirmation

Delivery confirmation for the message-data transfer shall be performed explicitly by means of a series of the message-data transmission on connectionless UDP channels. On the other hand, delivery confirmation by means of the message-data transmission on connection-oriented TCP communication channels is performed implicitly and the TCP protocol provides the function.

- Delivery confirmation is performed by Acknowledge response from the node receiving a confirm type message-data frame. Acknowledge response is the confirmation responded to the sending node by the receiving node upon reception of a confirm type message-data frame without error. Each node shall provide the function for Acknowledge response.
- The acknowledge response or ACK shall be conveyed by means of cyclic-data transmission, ie. a cyclic-data frame to be sent out of the receiving node shall contain the ACK to the sending node.
- The cyclic-data frame is sent out of the receiving node when the receiving node obtains the token, so that the cyclic-data frame can convey one or more ACKs which corresponds to the confirm type message-data frames received until the receiving node obtains the token and/or the receiving node sends out the cyclic-data frame with these ACKs.

NOTE The ACKs are conveyed by the final-cyclic-data frame in which at a maximum eight ACKs can be contained.

Figure 14 shows an example of a sequential diagram of exchanging ACKs for message-data transmissions on connectionless UDP channels among node #1, node #2 and node #3.

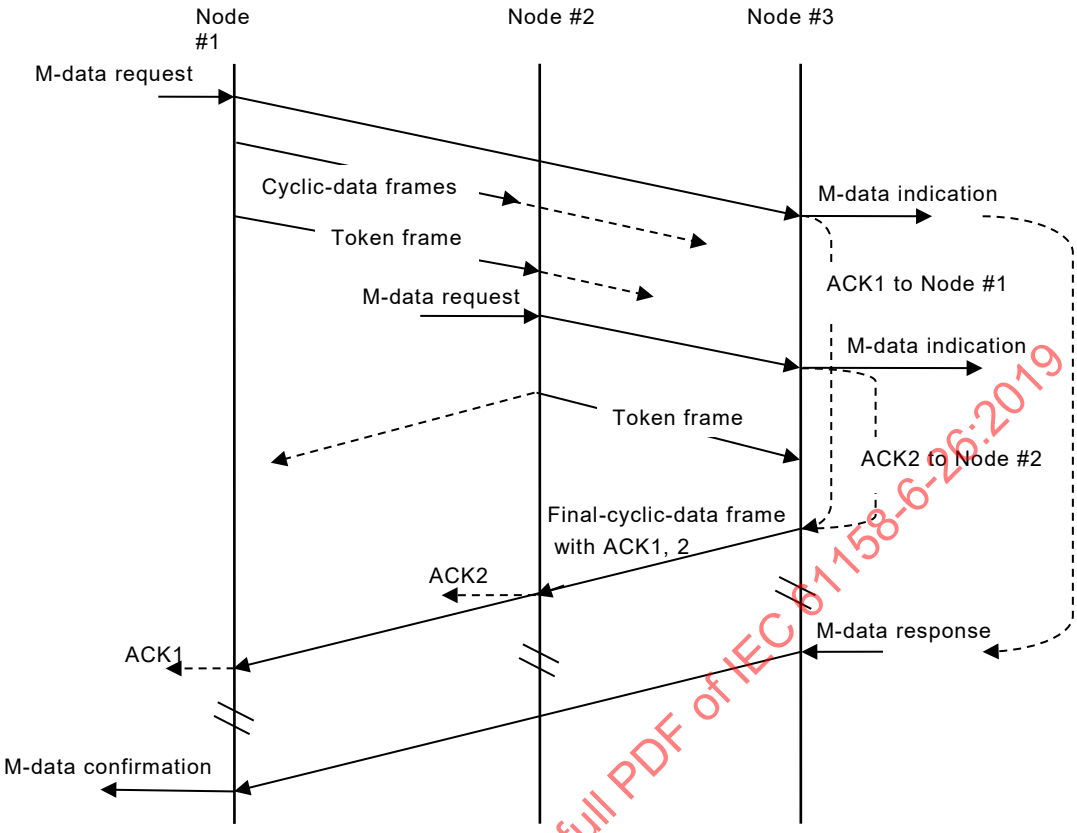


Figure 14 – Example of sequential diagram of ACK over UDP channel

Figure 15 shows the delivery confirmation or ACK sequence for a message-data transmission implicitly checked by TCP protocol on a connection-oriented TCP channel between node #1 and node #2.

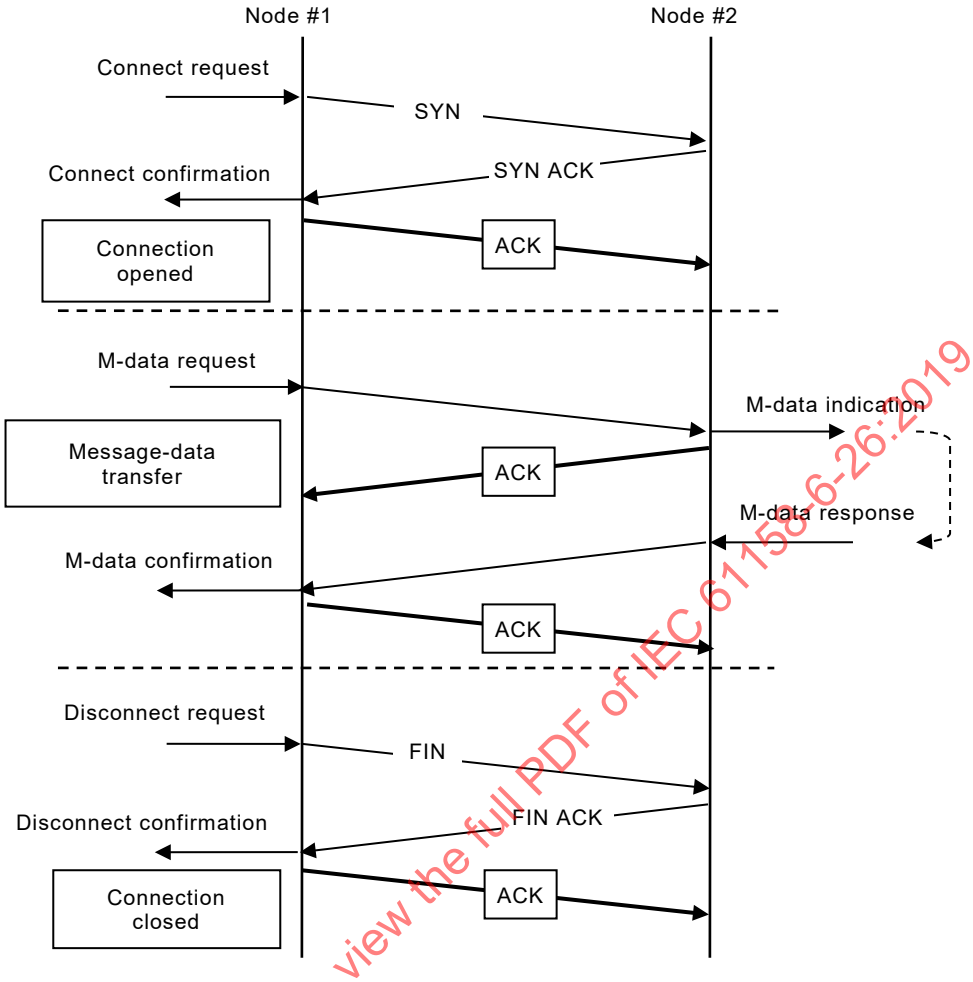


Figure 15 – Delivery confirmation checked by TCP protocol

4.3.6.3.5 Automatic retransmission

The automatic retransmission shall be performed by the nodes upon no ACK from the destination node of a message-data frame sent out and upon receiving ACK with error from the destination node, after sending out a message-data frame to the destination node.

Each node shall be provided with the automatic retransmission function as follows:

a) Upon reception of an ACK with error, the node shall perform the retransmission of the message-data transmission;

b) Upon the waiting-time-period-for-receiving-message-acknowledge timer (AWT-timer) out under no reception of any ACK from the destination node during the three-lap time period of the token circulation (3CWT time period) to the node after sending out a message-data frame, the node shall perform the retransmission of the message-data transmission;

c) Retransmission by the node shall be carried out at a maximum three times;

d) Upon no reception of any ACK from the destination node after retransmissions three times, the node shall cancel the message-data transmission and shall notify of the error result with error information to the FAL-user.

4.3.6.3.6 Sequence Number Management

The sequence number management is utilized to check and determine whether a receiving message-data frame has been previously received and is redundant.

The sequence number management by each node is as follows:

a) The sequence number management by each node shall be performed by means of both of the version of sequence number and the sequence number;

b) The value of the version of sequence number shall be set at the time when a node enters into the logical ring as a new node and shall be maintained until the node drops out of the logical ring, and is used for other nodes to determine whether a receiving message-data from the node is valid and not overdue;

c) The value of the version of sequence number shall be a unique number and is supposed to be generated by each node, for instance from current time data, random number data or random data and so forth. However the method to generate the unique number is implementation matter and is intentionally excluded in this document;

d) The value of the sequence number is used for other nodes to determine whether a receiving message-data from the node has been previously received and is redundant, and shall be set to zero (0) at the time of the network start-up, and shall be increased by one (1) every time on a message-data transmission completed; i.e. the sequence number shall be increased by one upon completion of a message-data transmission with ACK reception without error or completion with retransmission over due to error and then cancelling the message-data transmission; the value of the sequence number shall be a roll-over binary counter value of 32-bit length in the range of 0x1 to 0xFFFF FFFF;

e) The version of sequence number and the sequence number shall be set in the FA-link header and shall be conveyed by means of the cyclic-data and the message-data transmission to other nodes;

f) The values of the version of sequence number and the sequence number which are conveyed by a message-data transmission from other nodes shall be put into the corresponding entry area to each node in the message-sequence-number-management-information on one node to node basis upon reception of a message-data frame from a node;

g) When the value of the version of sequence number in the FA-link header of received message-data frame differs from the value of the corresponding node in the message-sequence-number-management-information, the value in the message-sequence-number-management-information shall be updated to the value in the FA-link header; and the received message-data frame shall be discarded; and ACK with error status of the version of sequence number different shall be sent back to the initiator node or the source node of the message-data frame;

h) The contents of the message-sequence-number-management-information shall be cleared in advance of each node entering into the logical ring;

i) When the value of the version of the sequence number of the corresponding node in the message-sequence-number-management-information is zero (0) as its initial value, the receiving message-data frame shall be valid without check; and the value of the version of the sequence number in the FA-link header of a received message-data frame shall be put into the corresponding entry area to that node in the message-sequence-number-management-information;

j) When the value of the sequence number in the FA-link header of the receiving message-data frame is equal to the value of the sequence number of the corresponding node in the message-sequence-number-management-information, the node receiving the message-data frame shall recognize that the receiving message-data frame is being retransmitted by the corresponding node; and the received message-data is already processed and then the receiving message-data frame shall be discarded; and the ACK without error status shall be sent back to the corresponding node;

k) When the value of the sequence number in the FA-link header of the receiving message-data frame differs from the value of the sequence number of the corresponding node in the message-sequence-number-management-information, the node receiving the message-data frame shall recognize the receiving message-data as a new one and shall put the value of received sequence number into the corresponding entry area to that node in the message-sequence-number-management-information.

4.3.7 Data transmission frames

4.3.7.1 Data transfer frame

Operation in normal, steady state on the node consists of a data transmission phase and a token transmission phase after a node obtains the token.

In the data transmission phase there exists several patterns of the cyclic-data transmission and the message-data transmission to be carried out by a node as follows:

a) When a node has no cyclic-data to send out due to no allocation of the memory area specific for this node on the common-memory and has a request to send a message-data from the FAL-user, upon obtaining the token the node first sends out a message-data frame and in succession sends out a cyclic-data frame with no cyclic-data. After sending out the cyclic-data frame without the cyclic-data, then the node sends out a token frame to pass the token to the next node on a logical ring;

b) When a node has cyclic-data of which data volume is less than or equal to 1 024 octets and has a request to send a message-data from the FAL-user, upon obtaining the token the node first sends out a message-data frame and in succession sends out one cyclic-data frame with the cyclic-data, then the node sends out a token frame to pass the token to the next node on a logical ring;

c) When a node has cyclic-data of which data volume is greater than 1 024 octets and has a request to send a message-data from the FAL-user, upon obtaining the token the node first sends out a message-data frame and in succession sends out a series of the cyclic-data frames with fragmented cyclic-data. After sending out the final-cyclic-data frame, the node sends out a token frame to pass the token to the next node on a logical ring.

Figure 16 shows the train of data frames and a token frame sending out of a node.

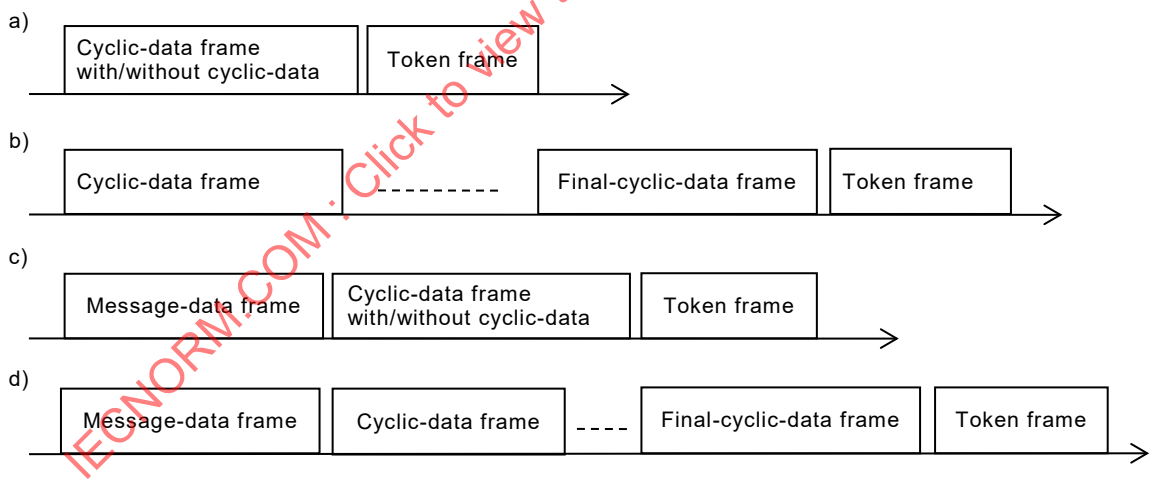
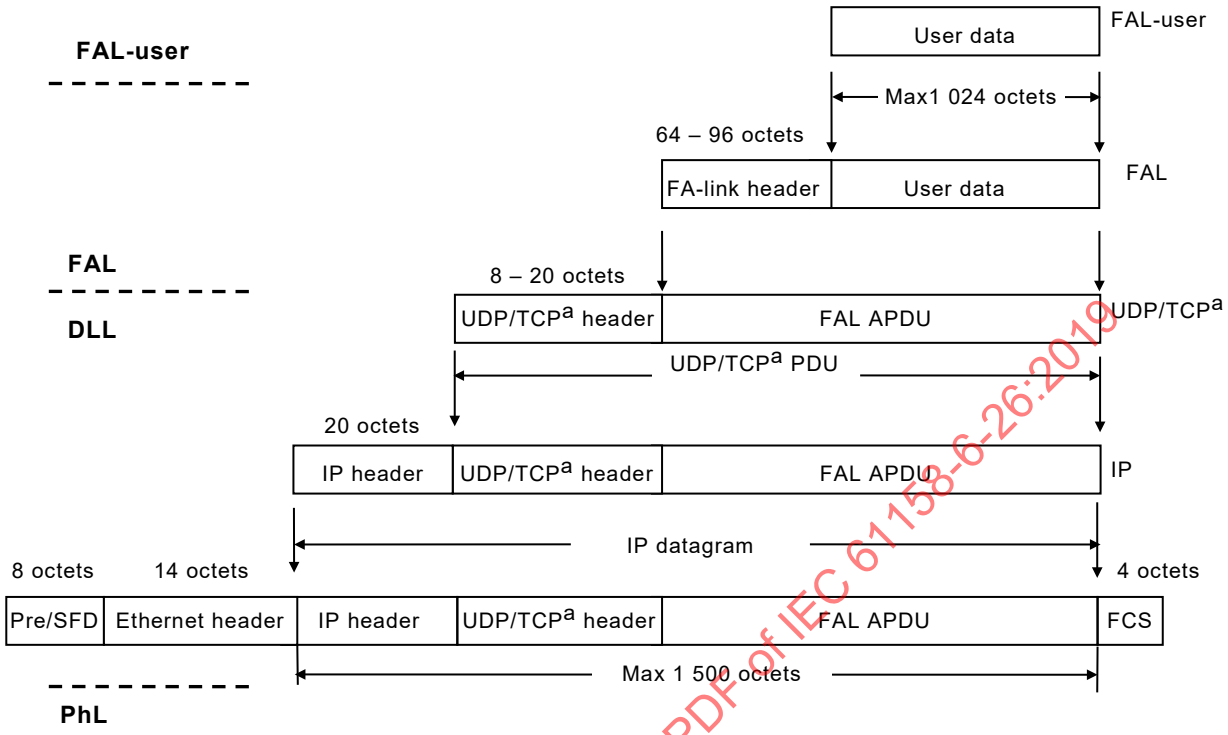


Figure 16 – Train of data frames and a token frame

4.3.7.2 Frame structure

Figure 17 shows the frame structure sent out on Ethernet or ISO/IEC/IEEE 8802-3 medium.



[a] Message-data transmission is performed primarily by means of connectionless UDP communication channels. The message-data transmission based on connection-oriented TCP communication channels is only for the General-purpose-command-server-communication which is utilized for intercommunication between the nodes and the general-purpose Human-Machine-Interface tool.

Figure 17 – Frame structure

4.3.7.3 Summary of data transmission frames

Table 4 summarises the data transmission frames including the channel control frames for Type 26 fieldbus.

Transaction code (TCD) is used to discriminate and identify each data transmission frame, and represents the transaction of a receiving frame. Several sets of the values and TCDs are uniquely defined and exclusively reserved for future use; however some of the values reserved are available for Type 26 application users uniquely without overlapping to define for their own application.

Table 4 – Data transmission frame and the TCD value

Data transmission frame	Value of TCD
Transparent-message	10 000 to 59 999 (‘2710’H..’EA5F’H)
Token	65 000 (‘FDE8’H)
Cyclic-data	65 001 (‘FDE9’H)
Participation-request (Participation-req)	65 002 (‘FDEA’H)
Byte-block-read request (Byte-block-read-req)	65 003 (‘FDEB’H)
Byte-block-read response (Byte-block-read-rsp)	65 203 (‘FEB3’H)
Byte-block-write request (Byte-block-write-req)	65 004 (‘FDEC’H)
Byte-block-write response (Byte-block-write-rsp)	65 204 (‘FEB4’H)
Word-block-read request (Word-block-read-req)	65 005 (‘FDED’H)

Data transmission frame	Value of TCD
Word-block-read response (Word-block-read-rsp)	65 205 (‘FEB5’H)
Word-block-write request (Word-block-write-req)	65 006 (‘FDEE’H)
Word-block-write response (Word-block-write-rsp)	65 206 (‘FEB6’H)
Network-parameter-read request (Network-parameter-read-req)	65 007 (‘FDEF’H)
Network-parameter-read response (Network-parameter-read-rsp)	65 207 (‘FEB7’H)
Network-parameter-write request (Network-parameter-write-req)	65 008 (‘FDF0’H)
Network-parameter-write response (Network-parameter-write-rsp)	65 208 (‘FEB8’H)
Stop-command request (Stop-command-req)	65 009 (‘FDF1’H)
Stop-command response (Stop-command-rsp)	65 209 (‘FEB9’H)
Operation-command request (Operation-command-req)	65 010 (‘FDF2’H)
Operation-command response (Operation-command-rsp)	65 210 (‘FEBA’H)
Profile-read request (Profile-read-req)	65 011 (‘FDF3’H)
Profile-read response (Profile-read-rsp)	65 211 (‘FEBB’H)
Trigger	65 012 (‘FDF4’H)
Log-data-read request (Log-data-read-req)	65 013 (‘FDF5’H)
Log-data-read response (Log-data-read-rsp)	65 213 (‘FEBD’H)
Log-data-clear request (Log-data-clear-req)	65 014 (‘FDF6’H)
Log-data-clear response (Log-data-clear-rsp)	65 214 (‘FEBE’H)
Message-return request (Message-return-req)	65 015 (‘FDF7’H)
Message-return response (Message-return-rsp)	65 215 (‘FEBF’H)
Vendor-specific-message request (Vendor-specific-msg-req)	65 016 (‘FDF8’H)
Vendor-specific-message response (Vendor-specific-msg-rsp)	65 216 (‘FEC0’H)
Start-TK-holding-time-measurement request (Start-TK-hld-time-mrmt-req)	65 020 (‘FDFC’H)
Start-TK-holding-time-measurement response (Start-TK-hld-time-mrmt-rsp)	65 220 (‘FEC4’H)
Terminate-TK-holding-time-measurement request (Terminate-TK-hld-time-mrmt-req)	65 021 (‘FDFD’H)
Terminate-TK-holding-time-measurement response (Terminate-TK-hld-time-mrmt-rsp)	65 221 (‘FEC5’H)
Start-GP_Comm-sender-log request (Start-GP_Comm-sndr-log-req)	65 022 (‘FDFE’H)
Start-GP_Comm-sender-log response (Start-GP_Comm-sndr-log-rsp)	65 222 (‘FEC6’H)
Terminate-GP_Comm-sender-log request (Terminate-GP_Comm-sndr-log-req)	65 023 (‘FDFF’H)
Terminate-GP_Comm-sender-log response (Terminate-GP_Comm-sndr-log-rsp)	65 223 (‘FEC7’H)
Set-remote-node-configuration-parameter request (Set-remote-node-config-para- req)	65 024 (‘FE00’H)
Set-remote-node-configuration-parameter response (Set-remote-node-config-para- rsp)	65 224 (‘FEC8’H)
Read-remote-participating-node-management-information-parameter request (Read-rmt-partici-node-mgt-info-para-req)	65 025 (‘FE01’H)
Read-remote-participating-node-management-information-parameter response (Read-rmt-partici-node-mgt-info-para-rsp)	65 225 (‘FEC9’H)
Read-remote-node-management-information-parameter request (Read-rmt-node-mgt-info-para-req)	65 026 (‘FE02’H)

Data transmission frame	Value of TCD
Read-remote-node-management-information-parameter response (Read-rmt-node-mgt-info-para-rsp)	65 226 (‘FECA’H)
Read-remote-node-setting-information-parameter request (Read-rmt-node-set-info-para-req)	65 027 (‘FE03’H)
Read-remote-node-setting-information-parameter response (Read-rmt-node-set-info-para-rsp)	65 227 (‘FECB’H)
Reset-node request (Reset-node-req)	65 028 (‘FE04’H)
Reset-node response (Reset-node-rsp)	65 228 (‘FECC’H)
NOTE 1 The following TCD numbers are reserved for user applications: 0 to 9 999, 60 000 to 64 999, 65 212, 65 217 and 65 400 to 65 535. NOTE 2 The following TCD numbers are for future system extension: 65 017, 65 018, 65 019, 65 029 to 65 202, 65 218, 65 219 and 65 229 to 65 399.	

4.4 FAL PDU abstract syntax

4.4.1 Basic abstract syntax

4.4.1.1 PDU definition

The definitions of Type 26 FAL PDUs are shown below.

```
FAL-PDU::= CHOICE {
    transparent-msg-PDU                [0]  Transparent-msg-PDU
    token-PDU                          [1]  Token-PDU
    participation-req-PDU              [2]  Participation-req-PDU
    byte-block-read-req-PDU            [3]  Byte-block-read-req-PDU
    byte-block-read-rsp-PDU            [4]  Byte-block-read-rsp-PDU
    byte-block-write-req-PDU           [5]  Byte-block-write-req-PDU
    byte-block-write-rsp-PDU           [6]  Byte-block-write-rsp-PDU
    word-block-read-req-PDU            [7]  Word-block-read-req-PDU
    word-block-read-rsp-PDU            [8]  Word-block-read-rsp-PDU
    word-block-write-req-PDU           [9]  Word-block-write-req-PDU
    word-block-write-rsp-PDU           [10] Word-block-write-rsp-PDU
    network-parameter-read-req-PDU     [11] Network-parameter-read-req-PDU
    network-parameter-read-rsp-PDU     [12] Network-parameter-read-rsp-PDU
    network-parameter-write-req-PDU    [13] Network-parameter-write-req-PDU
    network-parameter-write-rsp-PDU    [14] Network-parameter-write-rsp-PDU
    stop-command-req-PDU               [15] Stop-command-req-PDU
    stop-command-rsp-PDU               [16] Stop-command-rsp-PDU
    operation-command-req-PDU          [17] Operation-command-req-PDU
    operation-command-rsp-PDU          [18] Operation-command-rsp-PDU
    profile-read-req-PDU               [19] Profile-read-req-PDU
    profile-read-rsp-PDU               [20] Profile-read-rsp-PDU
    trigger-PDU                        [21] Trigger-PDU
    log-data-read-req-PDU              [22] Log-data-read-req-PDU
    log-data-read-rsp-PDU              [23] Log-data-read-rsp-PDU
    log-data-clear-req-PDU             [24] Log-data-clear-req-PDU
    log-data-clear-rsp-PDU             [25] Log-data-clear-rsp-PDU
```

```
        message-return-req-PDU                          [26]    Message-return-req-PDU
        message-return-rsp-PDU                          [27]    Message-return-rsp-PDU
        vendor-specific-msg-req-PDU                     [28]    Vendor-specific-msg-req-PDU
        vendor-specific-msg-rsp-PDU                     [29]    Vendor-specific-msg-rsp-PDU
        start-TK-hld-time-mrmt-req-PDU                  [30]    Start-TK-hld-time-mrmt-req-PDU
        start-TK-hld-time-mrmt-rsp-PDU                  [31]    Start-TK-hld-time-mrmt-rsp-PDU
        terminate-TK-hld-time-mrmt-req-PDU              [32]    Terminate-TK-hld-time-mrmt-req-PDU
        terminate-TK-hld-time-mrmt-rsp-PDU              [33]    Terminate-TK-hld-time-mrmt-rsp-PDU
        start-GP_Comm-sndr-log-req-PDU                  [34]    Start-GP_Comm-sndr-log-req-PDU
        start-GP_Comm-sndr-log-rsp-PDU                  [35]    Start-GP_Comm-sndr-log-rsp-PDU
        terminate-GP_Comm-sndr-log-req-PDU              [36]    Terminate-GP_Comm-sndr-log-req-PDU
        terminate-GP_Comm-sndr-log-rsp-PDU              [37]    Terminate-GP_Comm-sndr-log-rsp-PDU
        set-remote-node-config-para- req-PDU            [38]    Set-remote-node-config-para- req-PDU
        set-remote-node-config-para- rsp-PDU            [39]    Set-remote-node-config-para- rsp-PDU
        read-rmt-partici-node-mgt-info-para-req-PDU     [40]    Read-rmt-partici-node-mgt-info-para-req-PDU
        read-rmt-partici-node-mgt-info-para-rsp-PDU     [41]    Read-rmt-partici-node-mgt-info-para-rsp-PDU
        read-rmt-node-mgt-info-para-req-PDU             [42]    Read-rmt-node-mgt-info-para-req-PDU
        read-rmt-node-mgt-info-para-rsp-PDU             [43]    Read-rmt-node-mgt-info-para-rsp-PDU
        read-rmt-node-set-info-para-req-PDU             [44]    Read-rmt-node-set-info-para-req-PDU
        read-rmt-node-set-info-para-rsp-PDU             [45]    Read-rmt-node-set-info-para-rsp-PDU
        reset-node-req-PDU                              [46]    Reset-node-req-PDU
        reset-node-rsp-PDU                              [47]    Reset-node-rsp-PDU
        cyclic-data-PDU                                         Cyclic-data-PDU
}

Cyclic-data-PDU::= CHOICE {
        cyclic-data-wo-ACK-PDU                          [48]    Cyclic-data-wo-ACK-PDU
        cyclic-data-w-ACK-PDU                           [49]    Cyclic-data-w-ACK-PDU
}
```

4.4.1.2 FALARHeader definition

FALARHeader to be used in each PDU are shown as follows.

```
FALARHeader::= SEQUENCE {
        h_type                  H_type
        tFL                     TFL
        sNA                     SNA
        dNA                     DNA
        v_Seq                   V_Seq
        seq                     Seq
        m_Ctl                   M_Ctl
        uLS                     ULS
        m_SZ                    M_SZ
        m_ADD                   M_ADD
        mFT                     MFT
        m_RLT                   M_RLT
```

reserved	Unsigned16
tCD	TCD
vER	VER
c_AD1	C_AD1
c_SZ1	C_SZ1
c_AD2	C_AD2
c_SZ2	C_SZ2
mode	Mode
p_Type	P_Type
pRI	PRI
cBN	CBN
tBN	TBN
bSIZE	BSIZE
lKS	LKS
tW	TW
rCT	RCT

}

4.4.2 Transparent-msg- PDU

Transparent-msg-PDU::= SEQUENCE {

fALARHeader	FALARHeader
data	Trans-msgData

}

4.4.3 Token-PDU

Token-PDU::= SEQUENCE {

fALARHeader	FALARHeader

}

4.4.4 Participation-req-PDU

Participation-req-PDU::= SEQUENCE {

fALARHeader	FALARHeader
nDN	NDN
vDN	VDN
mSN	MSN
reserved	Unsigned16

}

4.4.5 Byte-block-read PDUs

Byte-block-read-req-PDU::= SEQUENCE {

fALARHeader	FALARHeader

}

```
Byte-block-read-rsp-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
        data                            B_Blk_Rd_rspData
}
```

4.4.6 Byte-block-write PDUs

```
Byte-block-write-req-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
        data                            B_Blk_Wt_reqDat
}

Byte-block-write-rsp-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
        data                            B_Blk_Wt_rspData
}
```

4.4.7 Word-block-read PDUs

```
Word-block-read-req-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
}

Word-block-read-rsp-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
        data                            W_Blk_Rd_rspData
}
```

4.4.8 Word-block-write PDUs

```
Word-block-write-req-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
        data                            W_Blk_Wt_reqDat
}

Word-block-write-rsp-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
        data                            W_Blk_Wt_rspData
}
```

4.4.9 Network-parameter-read PDUs

```
Network-parameter-read-req-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
}
```

```
Network-parameter-read-rsp-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
        data                                Net-para-Rd-rspData
}
```

4.4.10 Network-parameter-write PDUs

```
Network-parameter-write-req-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
        data                                Net-para-Wrt-reqData
}

Network-parameter-write-rsp-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
        data                                Net-para-Wrt-rspData
}
```

4.4.11 Stop-command PDUs

```
Stop-command-req-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
}

Stop-command-rsp-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
        data                                Stop-cmdData
}
```

4.4.12 Operation-command PDUs

```
Operation-command-req-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
}

Operation-command-rsp-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
        data                                Op-cmdData
}
```

4.4.13 Profile-read PDUs

```
Profile-read-req-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
}
```

```
Profile-read-rsp-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
        data                            Profile-readData
}
```

4.4.14 Trigger-PDU

```
Trigger-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
        nDN                             NDN
        vDN                             VDN
        mSN                             MSN
        reserved                        Unsigned16
}
```

4.4.15 Log-data-read PDUs

```
Log-data-read-req-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
}

Log-data-read-rsp-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
        data                            Log-readData
}
```

4.4.16 Log-data-clear PDUs

```
Log-data-clear-req-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
}

Log-data-clear-rsp-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
        data                            Log-clearData
}
```

4.4.17 Message-return PDUs

```
Message-return-req-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
        data                            Msg-return-reqData
}
```

```
Message-return-rsp-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
        data                            Msg-return-rspData
}
```

4.4.18 Vendor-specific-msg PDUs

```
Vendor-specific-msg-req-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
        vDN                             VDN
        sCODE                           SCODE
        data                            V_msg_reqData
}

Vendor-specific-msg-rsp-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
        vDN                             VDN
        sCODE                           SCODE
        data                            V_msg_rspData
}
```

4.4.19 Start-TK-hld-time-mrmt PDUs

```
Start-TK-hld-time-mrmt-req-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
}

Start-TK-hld-time-mrmt-rsp-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
}
```

4.4.20 Terminate-TK-hld-time-mrmt PDUs

```
Terminate-TK-hld-time-mrmt-req-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
}

Terminate-TK-hld-time-mrmt-rsp-PDU::= SEQUENCE {
        fALARHeader                     FALARHeader
        data                            TK-hld-timeData
}
```

4.4.21 Start-GP_Comm-sndr-log PDUs

Start-GP_Comm-sndr-log-req-PDU::= SEQUENCE {

fALARHeader	FALARHeader

}

Start-GP_Comm-sndr-log-rsp-PDU::= SEQUENCE {

fALARHeader	FALARHeader

}

4.4.22 Terminate-GP_Comm-sndr-log PDUs

Terminate-GP_Comm-sndr-log-req-PDU::= SEQUENCE {

fALARHeader	FALARHeader

}

Terminate-GP_Comm-sndr-log-rsp-PDU::= SEQUENCE {

fALARHeader	FALARHeader
data	Sndr-logData

}

4.4.23 Set-remote-node-config-para PDUs

Set-remote-node-config-para- req-PDU::= SEQUENCE {

fALARHeader	FALARHeader
data	Set-remote-node-config-para-ReqData

}

Set-remote-node-config-para- rsp-PDU::= SEQUENCE {

fALARHeader	FALARHeader
data	Set-remote-node-config-para-RspData

}

4.4.24 Read-rmt-partici-node-mgt-info-para PDUs

Read-rmt-partici-node-mgt-info-para-req-PDU::= SEQUENCE {

fALARHeader	FALARHeader
data	Read-rmt-partici-node-mgt-info-ReqData

}

Read-rmt-partici-node-mgt-info-para-rsp-PDU::= SEQUENCE {

fALARHeader	FALARHeader
data	Read-rmt-partici-node-mgt-info-RspData

}

4.4.25 Read-rmt- node-mgt-info-para PDUs

```
Read-rmt- node-mgt-info-para-req-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
}

Read-rmt- node-mgt-info-para-rsp-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
        data                                Rmt- node-mgt-info-paraData
}
```

4.4.26 Read-rmt-node-set-info-para PDUs

```
Read-rmt-node-set-info-para-req-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
}

Read-rmt-node-set-info-para-rsp-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
        data                                Set-info-para-read-data
}
```

4.4.27 Reset-node PDUs

```
Reset-node-req-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
}

Reset-node-rsp-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
}
```

4.4.28 Cyclic-data PDUs

```
Cyclic-data-wo-ACK-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
        cyclicData                          CyclicData
}

Cyclic-data-w-ACK-PDU::= SEQUENCE {
        fALARHeader                         FALARHeader
        aCKdata                             ACKdata
        cyclicData                          CyclicData
}
```

4.5 Data type assignments

Data types used in FAL PDU abstract syntax notation are shown as follows.

NOTE The data types are defined in 5.3 of IEC 61158-5-26.

H_type::= PrintableString SIZE (4)
TFL::= Unsigned32
SNA::= Unsigned32
DNA::= Unsigned32
V_Seq::= Unsigned32
SEQ::= Unsigned32
M_Ctl::= Unsigned32
ULS::= Unsigned16
M_SZ::= Unsigned16
M_ADD::= Unsigned32
MFT::= Unsigned8
M_RLT::= Unsigned8
TCD::= Unsigned16
VER::= Unsigned16
C_AD1::= Unsigned16
C_SZ1::= Unsigned16
C_AD2::= Unsigned16
C_SZ2::= Unsigned16
Mode::= Unsigned16
P_Type::= Unsigned8
PRI ::= Unsigned8
CBN::= Unsigned8
TBN::= Unsigned8
BSIZE::= Unsigned16
LKS::= Unsigned8
TW::= Unsigned8
RCT::= Unsigned16
NDN::= OctetString SIZE(10)
VDN::= OctetString SIZE(10)
MSN::= OctetString SIZE(10)
SCODE::= OctetString SIZE(6)
B_Blk_Rd_rspData::= OctetString SIZE(0..1 024)
B_Blk_Wt_reqDat::= OctetString SIZE(1..1 024)
B_Blk_Wt_rspData::= OctetString SIZE(0..1 024)
W_Blk_Rd_rspData::= OctetString SIZE(0..1 024)
W_Blk_Wt_reqData::= OctetString SIZE(2..1 024)
W_Blk_Wt_rspData::= OctetString SIZE(0..1 024)
Trans-msgData::= OctetString SIZE(0..1 024)
Net-para-Rd-rspData::= OctetString SIZE(0..1 024)
Net-para-Wrt-reqData::= OctetString SIZE(20)
Net-para-Wrt-rspData::= OctetString SIZE(0..1 024)
Stop-cmdData::= OctetString SIZE(0..1 024)
Op-cmdData::= OctetString SIZE(0..1 024)
Profile-readData::= OctetString SIZE(6..1 024)

Log-readData::= OctetString SIZE(0..512)

Log-clearData::= OctetString SIZE(0..512)

Msg-return-reqData::= OctetString SIZE(0..1 024)

Msg-return-rspData::= OctetString SIZE(0..1 024)

V_msg_reqData::= OctetString SIZE(0..1 024)

V_msg_rspData::= OctetString SIZE(0..1 024)

TK-hld-timeData::= OctetString SIZE(76)

Sndr-logData::= OctetString SIZE(84)

Set-remote-node-config-para-ReqData::= OctetString SIZE(28)

Set-remote-node-config-para-RspData::= OctetString SIZE(24)

Read-rmt-partici-node-mgt-info-ReqData::= OctetString SIZE(4)

Read-rmt-partici-node-mgt-info-RspData::= OctetString SIZE(20)

Rmt- node-mgt-info-paraData::= OctetString SIZE(64)

Set-info-para-read-data::= OctetString SIZE(96)

CyclicData::= OctetString SIZE (0..1 024)

ACKdata::= OctetString SIZE (20..132)

5 Transfer syntax

5.1 Encoding rules

5.1.1 Basic encoding

The encoding rule of this document is based on the terms and definitions of ISO/IEC 8825-1, which consists of the three encoding components; i.e. Identifier octet, Length octets and Contents octets. Using fixed length data, Identifier octet and Length octets like TER (Traditional Encoding Rule) are not used in this document.

5.1.2 Fixed length Unsigned encoding

The encoding of a fixed-length unsigned value of Unsigned8, Unsigned16 and Unsigned32 types shall be primitive as unsigned integers of one octet, two octets and four octets in length respectively. Unsigned8, Unsigned16 and Unsigned32 are encoded as big endians in which the most significant octet is regarded as the first octet, followed by the next octet as the second octet and the least significant octet is regarded as the last octet of the Contents octets.

5.1.3 Fixed length BitString encoding

The encoding of a fixed-length BitString value of BitString8, BitString16, and BitString32 types shall be primitive as unsigned integers of one octet, two octets and four octets in length respectively.

5.1.4 OctetString encoding

OctetString which has variable length of octets shall be encoded octet by octet in sequential order.

The Contents octets shall be equal in value to the octets in the data value, in the order they appear in the data value and with the most significant bit of an octet of the data value aligned with the most significant bit of an octet of the Contents octets.

5.1.5 SEQUENCE encoding

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

SEQUENCE encoding is performed in sequence, starting from the initial element, and the identifiers and length used in ASN.1 are not present.

5.2 PDU elements encoding

5.2.1 FALARHeader

5.2.1.1 Encoding

All the APDUs have the common header called FALARHeader. Each element field of FALARHeader that is defined in Clause 5 is encoded as big endians.

5.2.1.2 H_type

This field indicates the Type 26 fieldbus or “FL-net controller network”.

The value is “FACN”.

5.2.1.3 TFL

This field indicates the total octet length including both fields of the FALARHead and the data before fragmenting each PDU. When the data field contains multiple ACKs, the length of the ACKs shall be included.

The value of the total octet length shall be in the most significant Bit 31 through Bit 16. Bit 15..0 shall be the value ‘0…0’B.

5.2.1.4 SNA

This field contains the source node number.

The range of the value shall be 1 to 254, and the value shall be in the most significant octet.

5.2.1.5 DNA

This field contains the destination node number.

The value of DNA shall be in the most significant octet. The value is dependent on the PDUs as follows:

- For the PTP message-data transmission; the range of the value shall be 1 to 254;
- For the Token frame, the value is the node number of which a node is the successor node for the token to be passed to; the range of the value shall be 1 to 254;
- For the cyclic-data transmission, the value is the node number of which a node is the successor node for the token to be passed to; the range of the value shall be 1 to 254;
- For the BCT message-data transmission, the value is ‘FF’H;
- For the participation-request frame and the trigger frame, the value is ‘FF’H.

5.2.1.6 V_Seq

This field contains the Version of sequence number.

5.2.1.7 SEQ

This field contains the Sequence number.

The range of the value shall be ‘1’H to ‘FFFF FFFF’H.

5.2.1.8 M_CtI

This field indicates whether the PDU is of 1-to-N broadcast (BCT) type or 1-to-1 (PPT or Peer-to-Peer) type, and further indicates whether the PDU contains ACK data or not.

The bits of this field are defined as follows:

Bit 0	BCT:	If “True”, the PDU is the “broadcast” type.
Bit 1	PPT:	If “True”, the PDU is the “Peer-to-Peer” type.
Bit 2	Reserved:	This field is reserved for future use. The value is ‘0’B.
Bit 3	RPL:	If “True”, the PDU contains ACK data.
Bit 4..31	Reserved:	This field is reserved for future use. The value is ‘0…0’B.

5.2.1.9 ULS

This field contains the Upper layer status.

The bits of this field are defined as follows:

Bit 0..11	U_ERR_CODE:	Error-identifier is set by the upper layer.
Bit 12	Reserved:	This field is reserved for future use. The value is ‘0’B.
Bit 13	Warning:	If “True”, warning error exists.
Bit 14	ALARM:	If “True”, alarming error exists.
Bit 15	RUN/STOP:	If “True”, the upper layer process is in operation; If “False”, the upper layer process is halted.

Table 5 shows the upper layer operating condition matrix.

Table 5 – Upper layer operating condition matrix

Error Information	Operating state	
	RUN	STOP
NORMAL	The "upper layer process" is in operation, and both of the cyclic-data- and the message-data-transmission are applicable.	The "upper layer process" is halted, and both of the cyclic-data- and the message-data-transmission are applicable.
WARNING	The "upper layer process" is in operation under an error condition, and both of the cyclic-data- and the message-data-transmission are applicable.	The "upper layer process" is halted under the error condition, and both of the cyclic-data- and the message-data-transmission are applicable.
ALARM	Both of the cyclic-data and the message-data-transmission are not applicable.	Both of the cyclic data and message data are not applicable.

5.2.1.10 M_SZ, M_ADD

M_SZ is the message-data size of the virtual-address-space. M_ADD is the offset address of the target virtual-address-space. These fields are used for Byte-block-read/-write and Word-block-read/-write PDUs.

The range of the value for the M_SZ and the M_ADD is ‘0’H to ‘FFFF FFFF’H.

5.2.1.11 MFT

This field contains the allowable-minimum-frame-interval-time for the message-data transmission.

The range of the value shall be 0 to 50, and the unit is 100 μs.

5.2.1.12 M_RLT

This field contains the message result by the responder.

The value is as follows:

- ‘0’H Success: Successfully completed;
- ‘1’H Failure: Invalid requested parameter.
  Error information is contained in the data filed;
- ‘2’H Failure: Service unimplemented.

5.2.1.13 TCD

Refer to Table 4 in 4.3.7.3.

5.2.1.14 VER

This field indicates the program version.

The value is ‘0’H.

5.2.1.15 C_AD1, C_SZ1, C_AD2 and C_SZ2

C_AD1 is the data-head-address on common-memory-area-1, and C_SZ1 is the data-size on common-memory-area-1. C_AD1 is used in combination with C_SZ1. C_AD2 is the data-head-address on common-memory-area-2, and C_SZ2 is the data-size on common-memory-area-2. C_AD2 is used in combination with C_SZ2.

- The range of the C_AD1 is ‘0’H to ‘1FF’H, and the range of C_SZ1 is 0 to 512;
- The range of the C_AD2 is ‘0’H to ‘1 FFF’H, and the range of C_SZ2 is 0 to 8 192.

5.2.1.16 Mode

This field indicates the FA-link protocol version; i.e. the major version and the minor version, and the status of the token mode. The version of the FA-link protocol is indicated in combination with the major- and the minor-version number.

The bits of this field are defined as follows:

Bit 0..3	Reserved:	This field is reserved for future use. The value is ‘0…0’B.
Bit 4..7	Min_VER:	The range of the value shall be 0 to 15.
Bit 8..11	MAJ_VER:	The value is 2 or 3.
Bit 12..14	Reserved:	This field is reserved for future use. The value is ‘0…0’B.
Bit 15	Token:	The value is 1.

5.2.1.17 P_Type

This field indicates the protocol type.

The value is ‘80’H.

5.2.1.18 PRI

This field indicates the message-priority.

The value is ‘0’H.

5.2.1.19 CBN, TBN and BSIZE

CBN, TBN and BSIZE contain the Current-fragment-block-number, the Total-fragment-block-number and the Current-fragment-block-size respectively.

When the data field contains multiple ACKs, the length of the ACKs shall be included in the BSIZE. In the case of no fragmentation, the value of the TFL and the BSIZE is same value.

The range of the CBN and the TBN is 1 to 255, and the range of the BSIZE is 64 to 1 088.

5.2.1.20 LKS

This field contains the link-status.

The bits of this field are defined as follows:

Bit 0..3	Reserved:	The value is ‘0…0’B.
Bit 4	Upper_Layer_inactive:	If “True”, the upper layer is inactive.
Bit 5	CM_Data_Valid:	If “True”, the data on the CM is valid.
Bit 6	CM_Set_up_Comp:	If “True”, the parameter set-up for the CM is completed.
Bit 7	Overlapped_CM_area:	If “True”, the CM address spaces are overlapped.

5.2.1.21 TW

This field contains the value of the Observation time period for the token circulation.

The range of the value shall be 1 to 255, and the unit is 1 ms.

5.2.1.22 RCT

This field contains the value of the Allowable-refresh-cycle-time.

The range of the value shall be 0 to 65 535, and the unit is 1 ms.

5.2.2 Transparent-msg PDU

5.2.2.1 PDU specific values

Transparent-msg-PDU specific values are given in Table 6.

Table 6 – Transparent-msg-PDU specific values

Field	Value
TFL	‘40’H to ‘440’H
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘2710’H..’EA5F’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘40’H to ‘440’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

5.2.2.2 Trans-msgData

Trans-msgData contains a user-defined message data to a target node. The structure of the Trans-msgData is shown in Figure 18.

The size of the Trans-msgData is 0 to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15
0	User-defined message data
2	
4	
⋱	
1 022	

Figure 18 – Structure of Trans-msgData

5.2.3 Token-PDU

Token-PDU specific values are given in Table 7.

Table 7 – Token-PDU specific values

Field	Value
TFL	‘40’H
SEQ	Not used; ‘0’H
M_CTL	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
M_RLT	Not used; ‘0’H
TCD	‘FDE8’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘40’H

5.2.4 Participation-req-PDU

5.2.4.1 PDU specific values

Participation-req -PDU specific values are given in Table 8.

Table 8 – Participation-req -PDU specific values

Field	Value
TFL	‘60’H
SEQ	Not used; ‘0’H
M_CTL	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
M_RLT	Not used; ‘0’H
TCD	‘FDEA’H
CBN	Not used; ‘0’H
TBN	Not used; ‘0’H
BSIZE	‘60’H
LKS	Not used; ‘0’H
RCT	Not used; ‘0’H

5.2.4.2 NDN

This field shows the Node name by the end user.

The value is ASCII character string (SIZE (10)). Data shall be encoded as big endians.

5.2.4.3 VDN

This field shows the Vendor code of the node.

The value is ASCII character string (SIZE (10)). Data shall be encoded as big endians.

5.2.4.4 MSN

This field shows the Manufacturer model name of the node.

The value is ASCII character string (SIZE (10)). Data shall be encoded as big endians.

5.2.5 Byte-block-read PDUs

5.2.5.1 PDU specific values

Byte-block-read-req-PDU specific values are given in Table 9.

Table 9 – Byte-block-read-req-PDU specific values

Field	Value
TFL	‘41’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
MFT	Not used; ‘0’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
TCD	‘FDEB’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

Byte-block-read-rsp-PDU specific values are given in Table 10.

Table 10 – Byte-block-read-rsp-PDU specific values

Field	Value
TFL	‘41’H to ‘440’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
MFT	Not used; ‘0’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
TCD	‘FEB3’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

5.2.5.2 B_Blk_Rd_rspData

5.2.5.2.1 B_Blk_Rd_rspData with M_RTL = 0

B_Blk_Rd_rspData with M_RLT = 0 contains the byte-block-data read out of a target node.

The structure of B_Blk_Rd_rspData is shown in Figure 19 in the case of M_RLT = 0 or "Success – Successfully completed".

The content is dependent on FAL-users. The size of the B_Blk_Rd_rspData is up to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 7
0	Byte-block-data read out
1	
2	
⋱	
1 023	

Figure 19 – Structure of B_Blk_Rd_rspData with M_RLT = 0

5.2.5.2.2 B_Blk_Rd_rspData with M_RTL = 1

B_Blk_Rd_rspData with M_RLT = 1 contains the Error information reported by a target node. The structure of the B_Blk_Rd_rspData is shown in Figure 20 in the case of M_RLT = 1 or "Failure – Invalid requested parameter".

The content of the error information is dependent on FAL-users. The size of the B_Blk_Rd_rspData is up to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 7
0	Error information
1	
2	
⋱	
1 023	

Figure 20 – Structure of B_Blk_Rd_rspData in case of M_RLT = 1

5.2.5.2.3 B_Blk_Rd_rspData with M_RLT = 2

In the case of M_RLT = 2 or "Failure – Service unimplemented", there is no B_Blk_Rd_rspData from the target node.

5.2.6 Byte-block-write PDUs

5.2.6.1 PDU specific values

Byte-block-write-req-PDU specific values are given in Table 11.

Table 11 – Byte-block-write-req-PDU specific values

Field	Value
TFL	‘41’H to ‘440’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
MFT	Not used; ‘0’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
TCD	‘FDEC’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

Byte-block-write-rsp-PDU specific values are given in Table 12.

Table 12 – Byte-block-write-rsp-PDU specific values

Field	Value
TFL	‘41’H to ‘440’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
MFT	Not used; ‘0’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
TCD	‘FEB4’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

5.2.6.2 B_Blk_Wt_reqData

B_Blk_Wt_reqData contains the byte-block-data to be written to a target node. The structure of B_Blk_Wt_reqData is shown in Figure 21.

The content is dependent on FAL-users. The size of the B_Blk_Wt_reqDat is up to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 7
0	Byte-block-data to be written
1	
2	
⋱	
1 023	

Figure 21 – Structure of B_Blk_Wt_reqDat

5.2.6.3 B_Blk_Wt_rspData

5.2.6.3.1 B_Blk_Wt_rspData with M_RTL = 0

In the case of M_RLT = 0 or "Success – Successfully completed", there is no B_Blk_Wt_rspData from the target node.

5.2.6.3.2 B_Blk_Wt_rspData with M_RTL = 1

B_Blk_Wt_rspData with M_RLT = 1 contains the Error information reported by a target node. The structure of the B_Blk_Wt_rspData is shown in Figure 22 in the case of M_RLT = 1 or "Failure – Invalid requested parameter".

The content of the error information is dependent on FAL-users. The size of the B_Blk_Rd_rspData is up to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 7
0	Error information
1	
2	
⋱	
1 023	

Figure 22 – Structure of B_Blk_Wt_rspData in case of M_RLT = 1

5.2.6.3.3 B_Blk_Wt_rspData with M_RLT = 2

In the case of M_RLT = 2 or "Failure – Service unimplemented", there is no B_Blk_Rd_rspData.

5.2.7 Word-block-read PDUs

5.2.7.1 PDU specific values

Word-block-read-req-PDU specific values are given in Table 13.

Table 13 – Word-block-read-req-PDU specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
MFT	Not used; ‘0’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
TCD	‘FDED’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

Word-block-read-rsp-PDU specific values are given in Table 14.

Table 14 – Word-block-read-rsp-PDU specific values

Field	Value
TFL	‘40’H to ‘440’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
MFT	Not used; ‘0’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
TCD	‘FEB5’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

5.2.7.2 W_Blk_Rd_rspData

5.2.7.2.1 W_Blk_Rd_rspData with M_RLT = 0

W_Blk_Rd_rspData with M_RLT = 0 contains the word-block-data read out of a target node.

The structure of W_Blk_Rd_rspData is shown in Figure 23 in the case of M_RLT = 0 or “Success – Successfully completed”.

The content is dependent on FAL-users. The size of the W_Blk_Rd_rspData is up to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15
0	Word-block-data read out
2	
4	
⋱	
1 022	

Figure 23 – Structure of W_Blk_Rd_rspData with M_RLT = 0

5.2.7.2.2 W_Blk_Rd_rspData with M_RLT = 1

W_Blk_Rd_rspData with M_RLT = 1 contains the Error information reported by a target node. The structure of the W_Blk_Rd_rspData is shown in Figure 24 in the case of M_RLT = 1 or "Failure – Invalid requested parameter".

The content of the error information is dependent on FAL-users. The size of the W_Blk_Rd_rspData is up to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15
0	Error information
2	
4	
⋱	
1 022	

Figure 24 – Structure of W_Blk_Rd_rspData in case of M_RLT = 1

5.2.7.2.3 W_Blk_Rd_rspData with M_RLT = 2

In the case of M_RLT = 2 or "Failure – Service unimplemented", there is no W_Blk_Rd_rspData from the target node.

5.2.8 Word-block-write PDUs

5.2.8.1 PDU specific values

Word-block-write-req-PDU specific values are given in Table 15.

Table 15 – Word-block-write-req-PDU specific values

Field	Value
TFL	‘42’H to ‘440’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
MFT	Not used; ‘0’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
TCD	‘FDEE’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

Word-block-write-rsp-PDU specific values are given in Table 16.

Table 16 – Word-block-write-rsp-PDU specific values

Field	Value
TFL	‘40’H to ‘440’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used; ‘0’H
MFT	Not used; ‘0’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
TCD	‘FEB6’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

5.2.8.2 W_Blk_Wt_reqDat

W_Blk_Wt_reqDat contains the word-block-data to be written to a target node. The structure of W_Blk_Wt_reqData is shown in Figure 25.

The content is dependent on FAL-users. The size of the W_Blk_Wt_reqDat is up to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 7
0	Word-block-data to be written
1	
2	
⋱	
1 023	

Figure 25 – Structure of W_Blk_Wt_reqDat

5.2.8.3 W_Blk_Wt_rspData

5.2.8.3.1 W_Blk_Wt_rspData with M_RLT = 0

In the case of M_RLT = 0 or "Success – Successfully completed", there is no W_Blk_Wt_rspData from the target node.

5.2.8.3.2 W_Blk_Wt_rspData with M_RLT = 1

W_Blk_Wt_rspData with M_RLT = 1 contains the Error information reported by a target node. The structure of the W_Blk_Wt_rspData is shown in Figure 26 in the case of M_RLT = 1 or "Failure – Invalid requested parameter".

The content of the error information is dependent on FAL-users. The size of the W_Blk_Wt_rspData is up to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15
0	Error information
2	
4	
⋱	
1 022	

Figure 26 – Structure of W_Blk_Wt_rspData in case of M_RLT = 1

5.2.8.3.3 W_Blk_Wt_rspData with M_RLT = 2

In the case of M_RLT = 2 or "Failure – Service unimplemented", there is no W_Blk_Wt_rspData from the target node.

5.2.9 Network-parameter-read PDUs

5.2.9.1 PDU specific values

Network-parameter-read-req-PDU specific values are given in Table 17.

Table 17 – Network-parameter-read-req-PDU specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FDEF’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘40’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

Network-parameter-read-rsp-PDU specific values are given in Table 18.

Table 18 – Network-parameter-read-rsp-PDU specific values

Field	Value
TFL	‘78’H to ‘440’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FEB7’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘78’H to ‘440’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

5.2.9.2 Net-para-Rd-rspData

5.2.9.2.1 Net-para-Rd-rspData with M_RLT = 0

Net-para-Rd-rspData with M_RLT = 0 contains the values of the network parameters read from a target node. The structure of the Net-para-Rd-rspData is shown in Figure 27 in the case of M_RLT = 0 or “Success – Successfully completed”.

The values of the data elements of the Net-para-Rd-rspData are shown in Table 19. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 7	Bit 8 Bit 15
0	NDN	
⋮		
8		
10	VDN	
⋮		
18		
20	MSN	
⋮		
28		
30	C_AD1	
32	C_SZ1	
34	C_AD2	
36	C_SZ2	
38	Reserved	TW
40	Reserved	MFT
42	Reserved	LKS
44	Reserved	P_Type
46	ULS	
48	RCT	
50	RMT_current	
52	RMT_maximum	
54	RMT_minimum	

Figure 27 – Structure of Net-para-Rd-rspData

Table 19 – Values of data elements of Net-para-Rd-rspData

Data element	Value	Description
NDN	ASCII character string (SIZE (10))	Node name by the end user
VDN	ASCII character string (SIZE (10))	Vender code of the node
MSN	ASCII character string (SIZE (10))	Manufacturer's model name of a Type 26 node
C_AD1	'0'H to '1FF'H	Data-head-address on common-memory-area-1
C_SZ1	0 to 512	Data-size on common-memory-area-1
C_AD2	'0'H to '1 FFF'H	Data-head-address on common-memory-area-1
C_SZ2	0 to 8 192	Data-size on common-memory-area-1
TW	1 to 255; The unit is 1 ms.	Observation time period for the token circulation
MFT	0 to 50; The unit is 100 μs.	Allowable-minimum-frame-interval-time for the message-data transmission
LKS	Refer to 5.2.1.20	Link-status
P_Type	'80'H	Protocol type
ULS	Refer to 5.2.1.9	Upper layer status
RCT	'0'H to 'FFFF'H; The unit is 1 ms.	Allowable-refresh-cycle-time
RMT_current	'0'H to 'FFFF'H; The unit is 1 ms.	Current value of the Refresh-cycle-measurement-time
RMT_maximum	'0'H to 'FFFF'H; The unit is 1 ms.	Maximum value of the Refresh-cycle-measurement-time
RMT_minimum	'0'H to 'FFFF'H; The unit is 1 ms.	Minimum value of the Refresh-cycle-measurement-time
Reserved	'0'H	Reserved for future use

5.2.9.2.2 Net-para-Rd-rspData with M_RLT = 1

Net-para-Rd-rspData with M_RLT = 1 contains the error information reported by a target node. The structure of the Net-para-Rd-rspData is shown in Figure 28 in the case of M_RLT = 1 or "Failure – Invalid requested parameter".

The content of the error information is dependent on the device vendor. The size of the data is up to 1 024 octets. Data shall be encoded as little endians.

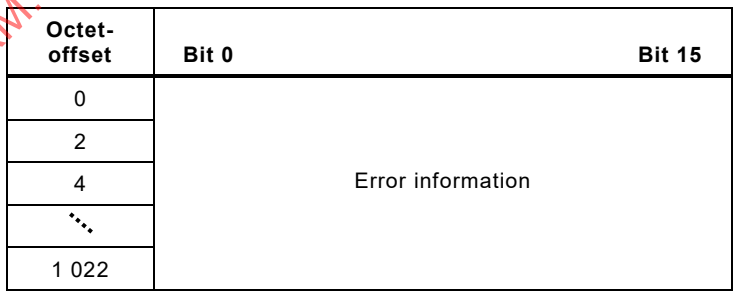


Figure 28 – Structure of Net-para-Rd-rspData with M_RLT = 1

5.2.10 Network-parameter-write PDUs

5.2.10.1 PDU specific values

Network-parameter-write-req-PDU specific values are given in Table 20.

Table 20 – Network-parameter-write-req-PDU specific values

Field	Value
TFL	‘54’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FDF0’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘54’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

Network-parameter-write-rsp-PDU specific values are given in Table 21.

Table 21 – Network-parameter-write-rsp-PDU specific values

Field	Value
TFL	‘40’H to ‘440’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FEB8’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H to ‘400’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

5.2.10.2 Net-para-Wrt-reqData

5.2.10.2.1 Structure of Net-para-Wrt-reqData

Net-para-Wrt-reqData contains the value of the network parameters to be set to the target node. The structure of the Net-para-Wrt-reqData is shown in Figure 29.

The values of the data elements of the Net-para-Wrt-reqData are shown in Table 22. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 7	Bit 8 Bit 15
0	Setting-para-flag	Reserved (‘0’H)
2	C_AD1	
4	C_SZ1	
6	C_AD2	
8	C_SZ2	
10	NDN	
⋱		
18		

Figure 29 – Structure of Net-para-Wrt-reqData

Table 22 – Values of the data elements of Net-para-Wrt-reqData

Data element	Value	Description
Setting-para-flag	'1'H: Set only both of the data-head-address and the data-size for CM area-1 and -2 '2'H: Set only the node name of the node (NDN) '3'H: Set all parameters	Options for setting parameters
C_AD1	‘0’H to ‘1FF’H	Data-head-address on common-memory-area-1
C_SZ1	0 to 512	Data-size on common-memory-area-1
C_AD2	‘0’H to ‘1 FFF’H	Data-head-address on common-memory-area-1
C_SZ2	0 to 8 192	Data-size on common-memory-area-1
NDN	ASCII character string (SIZE (10))	Node name by the end user

5.2.10.3 Net-para-Wrt-rspData

5.2.10.3.1 Net-para-Wrt-rspData with M_RLT = 0

In the case of M_RLT = 0 or “Success – Successfully completed”, there is no Net-para-Wrt-rspData.

5.2.10.3.2 Net-para-Wrt-rspData with M_RLT = 1

Net-para-Wrt-rspData with M_RLT = 1 contains the error information reported by a target node. The structure of the Net-para-Wrt-rspData is shown in Figure 30 in the case of M_RLT = 1 or “Failure – Invalid requested parameter”.

The content of the error information is dependent on the device vendor. The size of the data is up to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15
0	Error information
2	
4	
⋱	
1 022	

Figure 30 – Structure of Net-para-Wrt-rspData with M_RLT = 1

5.2.10.3.3 Net-para-Wrt-rspData with M_RLT = 2

In the case of M_RLT = 2 or “Failure – Service unimplemented”, there is no Net-para-Wrt-rspData.

5.2.11 Stop-command PDUs

5.2.11.1 PDU specific values

Stop-command-req-PDU specific values are given in Table 23.

Table 23 – Stop-command-req-PDU specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FDF1’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘40’H
LKS	Not used; ‘0’H

Field	Value
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

Stop-command-rsp-PDU specific values are given in Table 24.

Table 24 – Stop-command-rsp-PDU specific values

Field	Value
TFL	‘40’H to ‘440’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FEB9’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘40’H to ‘440’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

5.2.11.2 Stop-cmdData

5.2.11.2.1 Stop-cmdData with M_RLT = 0

In the case of M_RLT = 0 or “Success – Successfully completed”, there is no Stop-cmdData.

5.2.11.2.2 Stop-cmdData with M_RLT = 1

Stop-cmdData with M_RLT = 1 contains the error information reported by a target node. The structure of the Stop-cmdData is shown in Figure 31 in the case of M_RLT = 1 or “Failure – Invalid requested parameter”.

The content of the error information is dependent on the device vendor. The size of the data is up to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15
0	Error information
2	
4	
⋱	
1 022	

Figure 31 – Structure of Stop-cmdData with M_RLT = 1

5.2.11.2.3 Stop-cmdData with M_RLT = 2

In the case of M_RLT = 2 or “Failure – Service unimplemented”, there is no Stop-cmdData.

5.2.12 Operation-command PDUs

5.2.12.1 PDU specific values

Operation-command-req-PDU specific values are given in Table 25.

Table 25 – Operation-command-req-PDU specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FDF2’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘40’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

Operation-command-rsp-PDU specific values are given in Table 26.

Table 26 – Operation-command-rsp-PDU specific values

Field	Value
TFL	‘40’H to ‘440’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FEBA’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘40’H to ‘440’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

5.2.12.2 Op-cmdData

5.2.12.2.1 Op-cmdData with M_RLT = 0

I n the case of M_RLT = 0 or “Success – Successfully completed”, there is no Op-cmdData.

5.2.12.2.2 Op-cmdData with M_RLT = 1

Op-cmdData with M_RLT = 1 contains the error information reported by a target node. The structure of the Op-cmdData is shown in Figure 32 in the case of M_RLT = 1 or “Failure – Invalid requested parameter”.

The content of the error information is dependent on the device vendor. The size of the data is up to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15
0	Error information
2	
4	
⋱	
1 022	

Figure 32 – Structure of Op-cmdData with M_RLT = 1

5.2.12.2.3 Op-cmdData with M_RLT = 2

In the case of M_RLT = 2 or “Failure – Service unimplemented”, there is no Op-cmdData.

5.2.13 Profile-read PDUs

5.2.13.1 PDU specific values

Profile-read-req-PDU specific values are given in Table 27.

Table 27 – Profile-read-req-PDU specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FDF3’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘40’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

Profile-read-rsp-PDU specific values are given in Table 28.

Table 28 – Profile-read-rsp-PDU specific values

Field	Value
TFL	‘40’H to ‘440’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FEBB’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H

Field	Value
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘40’H to ‘440’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

5.2.13.2 Profile-readData

5.2.13.2.1 Profile-readData with M_RLT = 0

Profile-readData with M_RLT = 0 contains the device profile data of a target node. The structure of the Profile-readData is shown in Figure 33 in the case of M_RLT = 0 or “Success – Successfully completed”.

The content of the device profile data is dependent on the device vendor. The size of the data is 6 to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15
0	Contents of device profile
2	
4	
⋱	
1 022	

Figure 33 – Structure of Profile-readData with M_RLT = 0

NOTE Assumable contents of the device profile are as follows, but are out of the scope of this document:

System related items:

– Version of the device profile common specification;

– System parameter ID;

– System parameter revision number;

– System parameter revision date;

– Device type;

– Vendor name;

– Product name;

– Device-specific parameter ID;

Communication object related items:

– Information on resource object;

– Information on status and mode;

– Information on downloading and uploading;

– Implementation of message services;

– Information on the transparent message service;

– Information on security;

– Implementation of the log data read service;

Network parameter related items:

– Node name of the node (NDN);

– Vendor code of the node (VDN);

– Manufacturer model name of the node (MSN);

– Data-head-address of Common-memory-area 1 (C_AD1);

– Data size of Common-memory-area 1 (C_SZ1);

– Data-head-address of Common-memory-area 2 (C_AD2);

– Data size of Common-memory-area 2 (C_SZ2);

– TW;

– MFT;

– LKS;

– P_TYPE;

– ULS;

– RCT;

– RMT.

5.2.13.2.2 Profile-readData with M_RLT = 1

Profile-readData with M_RLT = 1 contains the error information reported by a target node. The structure of the Profile-readData is shown in Figure 34 in the case of M_RLT = 1 or "Failure – Invalid requested parameter".

The content of the error information is dependent on the device vendor. The size of the data is up to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0	Bit 15
0	Error information	
2		
4		
⋱		
1 022		

Figure 34 – Structure of Profile-readData with M_RLT = 1

5.2.14 Trigger-PDU

5.2.14.1 PDU specific values

Trigger-PDU specific values are given in Table 29.

Table 29 – Trigger-PDU specific values

Field	Value
TFL	‘60’H
SEQ	Not used. ‘0’H.
M_CTL	Not used. ‘0’H.
M_SZ	Not used. ‘0’H.
M_ADD	Not used. ‘0’H.
M_RLT	Not used. ‘0’H.
TCD	‘FDF4’H
CBN	Not used. ‘0’H.
TBN	Not used. ‘0’H.
BSIZE	‘60’H
LKS	Not used. ‘0’H.
RCT	Not used. ‘0’H.

5.2.14.2 NDN

Refer to 5.2.4.2.

5.2.14.3 VDN

Refer to 5.2.4.3.

5.2.14.4 MSN

Refer to 5.2.4.4.

5.2.15 Log-data-read PDUs

5.2.15.1 PDU specific values

Log-data-read-req-PDU specific values are given in Table 30.

Table 30 – Log-data-read-req-PDU U specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FDF5’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘40’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

Log-data-read-rsp-PDU specific values are given in Table 31.

Table 31 – Log-data-read-rsp-PDU specific values

Field	Value
TFL	‘40’H to ‘240’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FEBD’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘40’H to ‘240’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

5.2.15.2 Log-readData

5.2.15.2.1 Log-readData with M_RLT = 0

Log-readData contains the log data read from a node. The structure of the Log-readData with M_RLT = 0 is shown in Figure 35. The structure and the values of the data elements of the Log-readData are shown in Table 32.

The size of the Log-readData is 0 to 512 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 31
0	Contents of Log-readData
4	
8	
⋱	
508	

Figure 35 – Structure of Log-readData with M_RLT = 0

Table 32 – Contents of Log-readData

Octet-offset	Data element	Value	Usage
0	Total count of transmission over the TSAP	Roll-over binary counter value of 32-bit length	m
4	Total count of transmission error over the TSAP	Roll-over binary counter value of 32-bit length	m
8	Total count of transmission error over the Ethernet	Roll-over binary counter value of 32-bit length	o
12..20	Additional information 1	Related additional information	o
24	Total count of reception over the TSAP	Roll-over binary counter value of 32-bit length	m
28	Total count of reception error over the TSAP	Roll-over binary counter value of 32-bit length	m
32	Total count of reception error over the Ethernet	Roll-over binary counter value of 32-bit length	o
36..44	Additional information 2	Related additional information	o
48	Total count of the token sent out since the measurement starts	Roll-over binary counter value of 32-bit length	o
52	Total count of the cyclic-data frame sent out since the measurement starts	Roll-over binary counter value of 32-bit length	o
56	Total count of the one-to-one message-data frame sent out since the measurement starts	Roll-over binary counter value of 32-bit length	o
60	Total count of the one-to-n message-data frame sent out since the measurement starts	Roll-over binary counter value of 32-bit length	o
64..68	Additional information 3	Related additional information	o
72	Total count of the token frame received since the measurement starts	Roll-over binary counter value of 32-bit length	o
76	Total count of the cyclic-data frame received since the measurement start	Roll-over binary counter value of 32-bit length	o
80	Total count of the one-to-one message-data frame received since the measurement starts	Roll-over binary counter value of 32-bit length	o

Octet-offset	Data element	Value	Usage
84	Total count of the one-to-n message-data frame received since the measurement starts	Roll-over binary counter value of 32-bit length	o
88..92	Additional information 4	Related additional information	o
96	Total count of errors in receiving the cyclic-data frame since the measurement starts	Roll-over binary counter value of 32-bit length	m
100	Total count of the cyclic-data frame received with CM address size error since the measurement starts	Roll-over binary counter value of 32-bit length	o
104	Total count of the cyclic-data frame received with fragment number error since the measurement starts	Roll-over binary counter value of 32-bit length	o
108	Total count of the cyclic-data frame received with total fragment count error since the measurement starts	Roll-over binary counter value of 32-bit length	o
112	Elapsed time since the measurement starts when the cyclic-data frame with block size error received	sec. Roll-over binary counter value of 32-bit length. The unit is 1 s.	o
116	Elapsed time since the measurement starts when the error is detected in receiving cyclic-data frame	sec. Roll-over binary counter value of 32-bit length. The unit is 1 s.	o
120..140	Additional information 5	Related additional information	o
144	Total count of the message-data frame retransmitted since the measurement starts	Roll-over binary counter value of 32-bit length	m
148	Total count of the message-data frame retransmitted over the retry counts since the measurement starts	Roll-over binary counter value of 32-bit length	m
152	Elapsed time since the measurement starts when the maximum refresh-cycle time occurred last	sec. Roll-over binary counter value of 32-bit length. The unit is 1 s.	o
156..164	Additional information 6	Related additional information	o
168	Total count of errors in reception of the message-data frame since the measurement starts	Roll-over binary counter value of 32-bit length	m
172	Total count of the message-data frame received with sequence number error since the measurement starts	Roll-over binary counter value of 32-bit length	o
176	Total count of the message-data frame retransmitted with same sequence number since the measurement starts	Roll-over binary counter value of 32-bit length	o
180	Elapsed time since the measurement starts when receiving message-data frame is in error	sec. Roll-over binary counter value of 32-bit length. The unit is 1 s.	o
184..188	Additional information 7	Related additional information	o
192	Total count of the ACK error since the measurement starts	Roll-over binary counter value of 32-bit length	m
196	Total count of the ACK error with wrong version number of sequence number since the measurement starts	Roll-over binary counter value of 32-bit length	o
200	Total count of the ACK error with wrong sequence number since the measurement starts	Roll-over binary counter value of 32-bit length	o
204	Total count of the ACK error with wrong node number since the measurement starts	Roll-over binary counter value of 32-bit length	o
208	Total count of the ACK error with wrong TCD since the measurement starts	Roll-over binary counter value of 32-bit length	o
212..236	Additional information 8	Related additional information	o

Octet-offset	Data element	Value	Usage
240	Total count of the duplicated token frame error since the measurement start	Roll-over binary counter value of 32-bit length	m
244	Total count of the token discarded since the measurement starts	Roll-over binary counter value of 32-bit length	m
248	Total count of the token reissued since the measurement starts	Roll-over binary counter value of 32-bit length	m
252	Elapsed time since the measurement starts when the token is discarded	sec. Roll-over binary counter value of 32-bit length. The unit is 1 s.	o
256	Elapsed time since the measurement starts when the token reissued last	sec. Roll-over binary counter value of 32-bit length. The unit is 1 s.	o
260	Total count of the token holding time out since the measurement starts	Roll-over binary counter value of 32-bit length	o
264	Total count of the token-watchdog-time out since the measurement starts	Roll-over binary counter value of 32-bit length	o
268	Elapsed time since the measurement starts when the token watchdog time out occurred last	sec. Roll-over binary counter value of 32-bit length. The unit is 1 s.	o
272	Maximum time of the token holding time since the measurement starts	µs Roll-over binary counter value of 32-bit length. The unit is 1 µs.	o
276	Minimum time of the token holding time since the measurement starts	µs Roll-over binary counter value of 32-bit length. The unit is 1 µs.	o
280	Elapsed time since the measurement starts when the maximum token holding time occurred last	sec. Roll-over binary counter value of 32-bit length. The unit is 1 s.	o
284	Elapsed time in operation state	Roll-over binary counter value of 32-bit length. sec. The unit is 1 s.	o
288	Elapsed time since the measurement starts when the token holding time out occurred last	sec. Roll-over binary counter value of 32-bit length. The unit is 1 s.	o
292	Total count of being in waiting frame reception state	Roll-over binary counter value of 32-bit length	m
296	Total count of entering in participating state	Roll-over binary counter value of 32-bit length	m
300	Total count of entering in dropping out state of the node	Roll-over binary counter value of 32-bit length	m
304	Total count of entering in dropping out state with the token skipped	Roll-over binary counter value of 32-bit length	m
308	Total count of entering in dropping out state of other nodes	Roll-over binary counter value of 32-bit length	m
312	Elapsed time since the measurement starts of the token-holding-time measurement	sec. Roll-over binary counter value of 32-bit length. The unit is 1 s.	o
316	Total count of the token frames received during the measurement of the token-holding time	Roll-over binary counter value of 32-bit length	o
320..328	Reserved	'0'H	–
332	Measurement time elapsed since the sender log measurement starts	sec. Roll-over binary counter value of 32-bit length. The unit is 1 s.	o
336..364	Participation node related information	Related data on participation nodes	o
368	IP 1	Sender IP address 1	o

Octet-offset	Data element	Value	Usage
372	Number of the frame reception from IP 1	Roll-over binary counter value of 32-bit length	o
376	IP 2	Sender IP address 2	o
380	Number of the frame reception from IP 2	Roll-over binary counter value of 32-bit length	o
384	IP 3	Sender IP address 3	o
388	Number of the frame reception from IP 3	Roll-over binary counter value of 32-bit length	o
392	IP 4	Sender IP address 4	o
396	Number of the frame reception from IP 4	Roll-over binary counter value of 32-bit length	o
400	IP 5	Sender IP address5	o
404	Number of the frame reception from IP 5	Roll-over binary counter value of 32-bit length	o
408	IP 6	Sender IP address 6	o
412	Number of the frame reception from IP 6	Roll-over binary counter value of 32-bit length	o
416	IP 7	Sender IP address 7	o
420	Number of the frame reception from IP 7	Roll-over binary counter value of 32-bit length	o
424	IP 8	Sender IP address 8	o
428	Number of the frame reception from IP 8	Roll-over binary counter value of 32-bit length	o
432	IP 9	Sender IP address 9	o
436	Number of the frame reception from IP 9	Roll-over binary counter value of 32-bit length	o
440	IP 10	Sender IP address 10	o
444	Number of the frame reception from IP 10	Roll-over binary counter value of 32-bit length	o
448..508	Vendor specific items	Items dependent on implementation	-
NOTE In the usage column, "m" denotes "mandatory" and "o" denotes "optional".			

5.2.15.2.2 Log-readData with M_RLT = 1

Log-readData with M_RLT = 1 contains the error information from a node. The structure of the Log-readData with M_RLT = 1 is shown in Figure 36.

The error information is dependent on the vendor. The size of the error information is 0 to 512 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15
0	Contents of Log-readData with M_RLT = 1
2	
4	
⋱	
510	

Figure 36 – Structure of Log-readData with M_RLT = 1

5.2.16 Log-data-clear PDUs

5.2.16.1 PDU specific values

Log-data-clear-req-PDU specific values are given in Table 33.

Table 33 – Log-data-clear-req-PDU specific values

Field	Value
TFL	‘40’H
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FDF6’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘40’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

Log-data-clear-rsp-PDU specific values are given in Table 34.

Table 34 – Log-data-clear-rsp-PDU specific values

Field	Value
TFL	‘40’H to ‘240’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FEBE’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘40’H to ‘240’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

5.2.16.2 Log-clearData

5.2.16.2.1 Log-clearData with M_RLT = 0

In the case of M_RLT = 0 or “Success – Successfully completed”, there is no Log-clearData.

5.2.16.2.2 Log-clearData with M_RLT = 1

Log-clearData with M_RLT = 1 contains the error information reported by a target node. The structure of the Log-clearData is shown in Figure 37.

The error information is dependent on the vendor. The size of the error information is 0 to 512 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15
0	Error information
2	
4	
⋱	
510	

Figure 37 – Structure of Log-clearData

5.2.17 Message-return PDUs

5.2.17.1 PDU specific values

Message-return-req-PDU specific values are given in Table 35.

Table 35 – Message-return-req-PDU specific values

Field	Value
TFL	‘40’H to ‘440’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FDF7’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘40’H to ‘440’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

Message-return-rsp-PDU specific values are given in Table 36.

Table 36 – Message-return-rsp-PDU specific values

Field	Value
TFL	‘40’H to ‘440’H
PPT	‘1’B
RPL	Not used; ‘0’B
ULS	Not used; ‘0’H
M_SZ	Not used; ‘0’H
M_ADD	Not used; ‘0’H
MFT	Not used; ‘0’H
TCD	‘FEBF’H
C_AD1	Not used; ‘0’H
C_SZ1	Not used; ‘0’H
C_AD2	Not used; ‘0’H
C_SZ2	Not used; ‘0’H
CBN	‘1’H
TBN	‘1’H

Field	Value
BSIZE	‘40’H to ‘440’H
LKS	Not used; ‘0’H
TW	Not used; ‘0’H
RCT	Not used; ‘0’H

5.2.17.2 Msg-return-reqData

Msg-return-reqData contains the test data of a message-data transmission test to a target node.

The structure of the Msg-return-reqData is shown in Figure 38.

The content is dependent on the test conducted. The size of the test data is 0 to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15
0	Test data
2	
4	
⋱	
1 022	

Figure 38 – Structure of Msg-return-reqData

5.2.17.3 Msg-return-rspData

Msg-return-rspData with M_RLT = 0 contains the received test data and sent back to the initiating node of a message-data transmission test.

The structure of the Msg-return-rspData is shown in Figure 39.

The content is dependent on the test conducted. The size of the test data received is 0 to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15
0	Test data received
2	
4	
⋱	
1 022	

Figure 39 – Structure of Msg-return-rspData

5.2.18 Vendor-specific-msg PDUs

5.2.18.1 PDU specific values

Vendor-specific-msg-req-PDU specific values are given in Table 37.

Table 37 – Vendor-specific-msg-req-PDU specific values

Field	Value
TFL	‘50’H to ‘450’H
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FDF8’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
VER	‘0’H
CBN	‘1’H
TBN	‘1’H
BSIZE	‘50’H to ‘450’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

Vendor-specific-msg-rsp-PDU specific values are given in Table 38.

Table 38 – Vendor-specific-msg-rsp-PDU specific values

Field	Value
TFL	‘50’H to ‘450’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FEC0’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
VER	‘0’H
CBN	‘1’H

Field	Value
TBN	'1'H
BSIZE	'50'H to '450'H
LKS	Not used. '0'H
TW	Not used. '0'H
RCT	Not used. '0'H

5.2.18.2 VDN

This field contains the Vendor code of the node.

The value is ASCII character string (SIZE (10)) and the content is dependent on the vendor of the node. The value is encoded as big endian.

5.2.18.3 SCODE

This field contains the vendor specific subcode.

The data type is OctetString SIZE(6) and the content is dependent on the vendor. The value is encoded as big endian.

5.2.18.4 V_msg_reqData

V_msg_reqData contains the message to be exchanged. The structure of the V_msg_reqData is shown in Figure 40.

The content is dependent on FAL-users. The size of the Message data is 6 to 1 024 octets. Data shall be encoded as little endians.

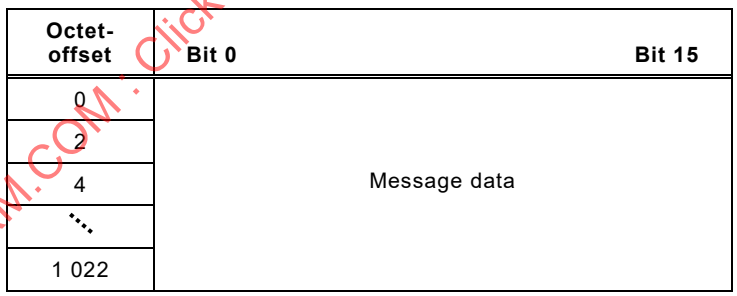


Figure 40 – Structure of V_msg_reqData

5.2.18.5 V_msg_rspData

5.2.18.5.1 V_msg_rspData with M_RLT = 0

V_msg_rspData contains the message data to be exchanged. The structure of the V_msg_rspData is shown in Figure 41 in the case of M_RLT = 0 or "Success – Successfully completed".

The content is dependent on FAL-users. The size of the V_msg_rspData is 0 to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15
0	Message data in case of M_RLT = 0
2	
4	
⋱	
1 022	

Figure 41 – Structure of V_msg_rspData in case of M_RLT = 0

5.2.18.5.2 V_msg_rspData with M_RLT = 1

V_msg_rspData with M_RLT = 1 contains the Error information reported by a target node. The structure of the V_msg_rspData is shown in Figure 42 in the case of M_RLT = 1 or “Failure – Invalid requested parameter”.

The content of the error information is dependent on FAL-users. The size of the V_msg_rspData is 0 to 1 024 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15
0	Error information
2	
4	
⋱	
1 022	

Figure 42 – Structure of V_msg_rspData in case of M_RLT = 1

5.2.18.5.3 V_msg_rspData with M_RLT = 2

In the case of M_RLT = 2 or “Failure – Service unimplemented”, there is no V_msg_rspData. When the value of the VDN in the Vendor-specific-msg-req-PDU is not available, the value of the M_RLT shall be set to 2.

5.2.19 Start-TK-hld-time-mrmt PDUs

Start-TK-hld-time-mrmt-req-PDU specific values are given in Table 39.

Table 39 – Start-TK-hld-time-mrmt-req-PDU specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FDFC’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

Start-TK-hld-time-mrmt-rsp-PDU specific values are given in Table 40.

Table 40 – Start-TK-hld-time-mrmt-rsp-PDU specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FEC4’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H

Field	Value
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

5.2.20 Terminate-TK-hld-time-mrmt PDUs

5.2.20.1 PDU specific values

Terminate-TK-hld-time-mrmt-req-PDU specific values are given in Table 41.

Table 41 – Terminate-TK-hld-time-mrmt-req-PDU specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FDFD’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

Terminate-TK-hld-time-mrmt-rsp-PDU specific values are given in Table 42.

Table 42 – Terminate-TK-hld-time-mrmt-rsp-PDU specific values

Field	Value
TFL	‘8C’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FEC5’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

5.2.20.2 TK-hld-timeData

The TK-hld-timeData contains the token-holding-time measurement result.

The structure of TK-hld-timeData is shown in Figure 43. The values of the data elements of TK-hld-timeData are shown in Table 43.

Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 31
0	TK_Discarded_count
4	TK_Discarded_time
8	TK_Reissued_count
12	TK_Reissued_time
16	TK-hld_timeout_count
20	TK-hld_timeout_time
24	TW_timeout_count
28	TW_timeout_t ime
32	Max_TK_hld_time
36	Min_TK_hld_time
40	Max_time
44	Mtime
48	TK_receive_count
52	Max_RFC_time
56	Cyclic_frame_count
60	Cyclic_frame_Error_count
64	Cyclic_frame_Error_time
68	Message_frame_count
72	Message_frame_Error_time

Figure 43 – Token-holding-time measurement result

Table 43 – Value of the data element of TK-hld-timeData

Data element	Value	Description
TK_Discarded_count	Roll-over binary counter value of 32-bit length	Total count of the token discarded since the measurement starts
TK_Discarded_time	The value range is 0 to 2^{31}. The unit is 1 s.	Elapsed time since the measurement starts when the token is discarded
TK_Reissued_count	Roll-over binary counter value of 32-bit length	Total count of the token reissued since the measurement starts
TK_Reissued_time	The value range is 0 to 2^{31}. The unit is 1 s.	Elapsed time since the measurement starts when the token reissued last
TK-hld_timeout_count	Roll-over binary counter value of 32-bit length	Total count of the token holding time out since the measurement starts
TK-hld_timeout_time	The value range is 0 to 2^{31}. The unit is 1 s.	Elapsed time since the measurement starts when the token holding time out occurred last
TW_timeout_count	Roll-over binary counter value of 32-bit length	Total count of the token-watchdog-time out since the measurement starts
TW_timeout_t ime	The value range is 0 to 2^{31}. The unit is 1 s.	Elapsed time since the measurement starts when the token watchdog time out occurred last
Max_TK_hld_time	The value range is 0 to 2^{31}. The unit is 1 μs.	Maximum time of the token holding time since the measurement starts
Min_TK_hld_time	The value range is 0 to 2^{31}. The unit is 1 μs.	Minimum time of the token holding time since the measurement starts

Data element	Value	Description
Max_time	The value range is 0 to 2^{31}. The unit is 1 s.	Elapsed time since the measurement starts when the maximum token holding time occurred last
Mtime	The value range is 0 to 2^{31}. The unit is 1 s.	Elapsed time since the measurement starts of the token-holding-time measurement
TK_receive_count	Roll-over binary counter value of 32-bit length	Total count of the token frames received during the measurement of the token-holding time
Max_RFC_time	The value range is 0 to 2^{31}. The unit is 1 s.	Elapsed time since the measurement starts when the maximum refresh-cycle time occurred last
Cyclic_frame_count	Roll-over binary counter value of 32-bit length	Total count of receiving the cyclic-data frame since the measurement starts
Cyclic_frame_Error_count	Roll-over binary counter value of 32-bit length	Total count of errors in receiving the cyclic-data frame since the measurement starts
Cyclic_frame_Error_time	Roll-over binary counter value of 32-bit length	Elapsed time since the measurement starts when the error is detected in receiving cyclic-data frame
Message_frame_count	Roll-over binary counter value of 32-bit length	Total count of errors in reception of the message-data frame since the measurement starts
Message_frame_Error_time	The value range is 0 to 2^{31}. The unit is 1 s.	Elapsed time since the measurement starts when receiving message-data frame is in error

5.2.21 Start-GP_Comm-sndr-log PDUs

Start-GP_Comm-sndr-log-req-PDU specific values are given in Table 44.

Table 44 – Start-GP_Comm-sndr-log-req-PDU specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FDFE’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H

Field	Value
RCT	Not used. ‘0’H

Start-GP_Comm-sndr-log-rsp-PDU specific values are given in Table 45.

Table 45 – Start-GP_Comm-sndr-log-rsp-PDU specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FEC6’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

5.2.22 Terminate-GP_Comm-sndr-log PDUs

5.2.22.1 PDU specific values

Terminate-GP_Comm-sndr-log-req-PDU specific values are given in Table 46.

Table 46 – Terminate-GP_Comm-sndr-log-req-PDU specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FDFF’H

Field	Value
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

Terminate-GP_Comm-sndr-log-rsp-PDU specific values are given in Table 47.

Table 47 – Terminate-GP_Comm-sndr-log-req-PDU specific values

Field	Value
TFL	‘94’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FEC7’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

5.2.22.2 Sndr-logData

The Sndr-logData contains the number of reception of the general-purpose-communication-data frames from the senders with the source IP addresses. The Sndr-logData is the log for the first 10 events of the reception from each sender.

The structure of Sndr-logData is shown in Figure 44. The values of the data elements of Sndr-logData are shown in Table 48.

The size of the Sndr-logData is 84 octets. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 31
0	Mtime
4	IP 1
8	Number of the frame reception from IP 1
12	IP 2
16	Number of the frame reception from IP 2
20	IP 3
24	Number of the frame reception from IP 3
28	IP 4
32	Number of the frame reception from IP 4
36	IP 5
40	Number of the frame reception from IP 5
44	IP 6
48	Number of the frame reception from IP 6
52	IP 7
56	Number of the frame reception from IP 7
60	IP 8
64	Number of the frame reception from IP 8
68	IP 9
72	I Number of the frame reception from IP 9
76	IP 10
80	Number of the frame reception from IP 10

Figure 44 – Structure of Sndr-logData

Table 48 – Value of the data element of Sndr-logData

Data element	Value	Description
Mtime	0 to 2^{31} The unit is 1 s.	Measurement time elapsed since the measurement starts
IP n	BitString32	Sender IP n address
Number of the frame reception from IP n	Roll-over binary counter value of 32-bit length	Total count of the frame reception from the source IP n address

5.2.23 Set-remote-node-config-para PDUs

5.2.23.1 PDU specific values

Set-remote-node-config-para-req-PDU specific values are given in Table 49.

Table 49 – Set-remote-node-config-para-req-PDU specific values

Field	Value
TFL	‘5C’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FE00’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

Set-remote-node-config-para-rsp-PDU specific values are given in Table 50.

Table 50 – Set-remote-node-config-para-rsp-PDU specific values

Field	Value
TFL	‘58’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FEC8’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H

Field	Value
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

5.2.23.2 Set-remote-node-config-para-ReqData

Set-remote-node-config-para-ReqData contains the values to be set to the node configuration parameters of a target node.

The structure of Set-remote-node-config-para-ReqData is shown in Figure 45. The values of the data elements of the Set-remote-node-config-para-ReqData are shown in Table 51. The bit definition of Update flag is shown in Table 52. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 7	Bit 8 Bit 15	Bit 16 Bit 23	Bit 24 Bit 31
0	Update flag		Reserved	
4	Node name			
8				
12			Reserved	
16	C_AD1		C_SZ1	
20	C_AD2		C_SZ2	
24	TW	Reserved	MFT	Reserved

Figure 45 – Structure of Set-remote-node-config-para-ReqData

Table 51 – Value of the data element of Set-remote-node-config-para-ReqData

Data element	Value	Description
Update flag	BitString16 Refer to Table 52.	Control for the update of each attribute
Node name	ASCII character string (SIZE (10))	Node name by the end user
C_AD1	‘0’H to ‘1FF’H	Data-head-address of Common-memory-area 1
C_SZ1	0 to 512	Data size of Common-memory-area 1
C_AD2	‘0’H to ‘1 FFF’H	Data-head-address of Common-memory-area 2
C_SZ2	0 to 8 192	Data size of Common-memory-area 2
TW	1 to 255; The unit is 1 ms.	Observation time period for the token circulation
MFT	0 to 50; The unit is 100 μs.	Allowable-minimum-frame-interval-time for the message-data transmission
Reserved	‘0’H	Reserved for future use

Table 52 – Bit definition of Update flag

Bit assignment	Data element	Description
Bit 0	Update C_AD & C_SZ	If “True”, update the C_AD and C_SZ1 values.
Bit 1	Update-Node name	If “True”, update the node name.
Bit 2	Update-TW	If “True”, update the TW value.
Bit 3	Update-MFT	If “True”, update the MFT value.
Bit 4..15	Reserved	Reserved for future use. The value is ‘0…0’B.

5.2.23.3 Set-remote-node-config-para-RspData

Set-remote-node-config-para-RspData contains the parameter values completed to set up to the node configuration parameters of the target node.

The structure of the Set-remote-node-config-para-RspData is shown in Figure 46. Each field contains the corresponding updated value. The values of the data elements of the Set-remote-node-config-para-RspData are shown in Table 53. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 7	Bit 8 Bit 15	Bit 16 Bit 23	Bit 24 Bit 31
0	Node name			
4				
8			‘0’H	
12	C_AD1		C_SZ1	
16	C_AD2		C_SZ2	
20	TW	‘0’H	MFT	‘0’H

Figure 46 – Structure of Set-remote-node-config-para-RspData

Table 53 – Value of the data element of Set-remote-node-config-para-RspData

Data element	Value	Description
Node name	ASCII character string (SIZE (10))	Updated the node name by the end user
C_AD1	‘0’H to ‘1FF’H	Updated the data-head-address of Common-memory-area 1
C_SZ1	0 to 512	Updated the data size of Common-memory-area 1
C_AD2	‘0’H to ‘1 FFF’H	Updated the data-head-address of Common-memory-area 2
C_SZ2	0 to 8 192	Updated the data size of Common-memory-area 2
TW	1 to 255; The unit is 1 ms.	Updated the observation time period for the token circulation
MFT	0 to 50; The unit is 100 μs.	Updated the allowable-minimum-frame-interval-time for the message-data transmission

5.2.24 Read-rmt-partici-node-mgt-info-para PDUs

5.2.24.1 PDU specific values

Read-rmt-partici-node-mgt-info-para-req-PDU specific values are given in Table 54.

Table 54 – Read-rmt-partici-node-mgt-info-para-req-PDU specific values

Field	Value
TFL	‘44’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FE01’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

Read-rmt-partici-node-mgt-info-para-rsp-PDU specific values are given in Table 55.

Table 55 – Read-rmt-partici-node-mgt-info-para-rsp-PDU specific values

Field	Value
TFL	‘54’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FEC9’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H

Field	Value
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

5.2.24.2 Read-rmt-partici-node-mgt-info-ReqData

Read-rmt-partici-node-mgt-info-ReqData contains a target node number. The structure of the Read-rmt-partici-node-mgt-info-ReqData is shown in Figure 47. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15	Bit 16 Bit 31
0	Node number 1 to 254	Reserved (‘0’H)

Figure 47 – Structure of Read-rmt-partici-node-mgt-info-ReqData

5.2.24.3 Read-rmt-partici-node-mgt-info-RspData

5.2.24.3.1 Structure of Read-rmt-partici-node-mgt-info-RspData

Read-rmt-partici-node-mgt-info-RspData contains the values of the participating node management information parameters transferred from a target node.

The structure of the Read-rmt-partici-node-mgt-info-RspData is shown in Figure 48. The values of the data elements of he Read-rmt-partici-node-mgt-info-RspData are shown in Table 56. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15	Bit 16 Bit 31
0	Node number	ULS
4	C_AD1	C_SZ1
8	C_AD2	C_SZ2
12	RCT	TW
16	MFT	LKS

Figure 48 – Structure of Read-rmt-partici-node-mgt-info-RspData

Table 56 – Value of the data element of Read-rmt-partici-node-mgt-info-RspData

Data element	Value	Description
Node number	1 to 254	Node number of the node
ULS	Refer to 5.2.1.9	Upper layer status
C_AD1	‘0’H to ‘1FF’H	Data-head-address on common-memory-area-1
C_SZ1	0 to 512	Data-size on common-memory-area-1
C_AD2	‘0’H to ‘1 FFF’H	Data-head-address on common-memory-area-1
C_SZ2	0 to 8 192	Data-size on common-memory-area-1
RCT	‘0’H to ’FFFF’H; The unit is 1 ms.	Allowable-refresh-cycle-time
TW	1 to 255; The unit is 1 ms.	Observation time period for the token circulation
MFT	0 to 50; The unit is 100 μs.	Allowable-minimum-frame-interval-time for the message-data transmission
LKS	Refer to 5.2.1.20	Link-status

5.2.25 Read-rmt- node-mgt-info-para PDUs

5.2.25.1 PDU specific values

Read-rmt- node-mgt-info-para-req-PDU specific values are given in Table 57.

Table 57 – Read-rmt- node-mgt-info-para-req-PDU specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FE02’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

Read-rmt- node-mgt-info-para-rsp-PDU specific values are given in Table 58.

Table 58 – Read-rmt- node-mgt-info-para-rsp-PDU specific values

Field	Value
TFL	‘80’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FECA’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

5.2.25.2 Rmt-node-mgt-info-paraData

Rmt-node-mgt-info-paraData contains the values of the node management information parameters transferred from a target node.

The structure of the Rmt-node-mgt-info-paraData is shown in Figure 49. The values of the data elements of the Rmt-node-mgt-info-paraData are shown in Table 59. The Bit definition of Node status is shown in Table 60. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15	Bit 16 Bit 31
0	Node number	Reserved
4	C_AD1	C_SZ1
8	C_AD2	C_SZ2
12	ULS	TW
16	MFT	Reserved
20	VDN	
24		
28		Reserved
32	MSN	
36		
40		Reserved
44	NDN	
48		
52		Reserved
56	P_TYPE	LKS
60	Node_status	Node_class

Figure 49 – Structure of Rmt-node-mgt-info-paraData

Table 59 – Value of the data element of Rmt-node-mgt-info-paraData

Data element	Value	Description
Node number	1 to 254	Node number of the node
C_AD1	‘0’H to ‘1FF’H	Data-head-address on common-memory-area-1
C_SZ1	0 to 512	Data-size on common-memory-area-1
C_AD2	‘0’H to ‘1 FFF’H	Data-head-address on common-memory-area-1
C_SZ2	0 to 8 192	Data-size on common-memory-area-1
ULS	Refer to 5.2.1.9	Upper layer status
TW	1 to 255; The unit is 1 ms.	Observation time period for the token circulation
VDN	ASCII character string (SIZE (10))	Vender code of the node
MSN	ASCII character string (SIZE (10))	Manufacturer's model name of a Type 26 node
NDN	ASCII character string (SIZE (10))	Node name by the end user
P_TYPE	‘80’H	Protocol type
LKS	Refer to 5.2.1.20	Link-status of the node
Node_status	BitString32 Refer to Table 60.	Node status of the node
Node_class	‘3’H	Node class identification

Table 60 – Bit definition of Node status

Bit assignment	Data element	Description
Bit 0..2	Reserved	Reserved for future use. The value is ‘0…0’B.
Bit 3	TW_timer_expired	If “True”, the TW_timer expired error happens.
Bit 4	Overlapped_CM_area	If “True”, the CM-address-area is overlapped.
Bit 5	Wait_for_recv	If “True”, the node is waiting for any reception.
Bit 6	Duplicate_Node	If “True”, the node number is duplicated.
Bit 7	Invalid_Initialize_Para	If “True”, the parameter set-up is completed.
Bit 8..15	Reserved	Reserved for future use. The value is ‘0…0’B.

5.2.26 Read-rmt-node-set-info-para PDUs

5.2.26.1 PDU specific values

Read-rmt-node-set-info-para-req-PDU specific values are given in Table 61.

Table 61 – Read-rmt-node-set-info-para-req-PDU specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FE03’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

Read-rmt-node-set-info-para-rsp-PDU specific values are given in Table 62.

Table 62 – Read-rmt-node-set-info-para-rsp-PDU specific values

Field	Value
TFL	‘58’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FECB’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

5.2.26.2 Set-info-para-read-data

Set-info-para-read-data contains the values of the node setting information parameters transferred from a target node.

The structure of the Set-info-para-read-data is shown in Figure 50. The Values of the data elements of the Set-info-para-read-data are shown in Table 63. Data shall be encoded as little endians.

Octet-offset	Bit 0 Bit 15	Bit 16 Bit 31
0	C_AD1	C_SZ1
4	C_AD2	C_SZ2
8	TW	MFT
12	NDN	
16		
20		‘0’H

Figure 50 – Structure of Set-info-para-read-data

Table 63 – Value of the data element of Set-info-para-read-data

Data element	Value	Description
C_AD1	‘0’H to ‘1FF’H	Data-head-address on common-memory-area-1
C_SZ1	0 to 512	Data-size on common-memory-area-1
C_AD2	‘0’H to ‘1 FFF’H	Data-head-address on common-memory-area-1
C_SZ2	0 to 8 192	Data-size on common-memory-area-1
TW	1 to 255; The unit is 1 ms.	Observation time period for the token circulation
MFT	0 to 50; T he unit is 100 μs.	Allowable-minimum-frame-interval-time
NDN	ASCII character string (SIZE (10))	Node name by the end user

5.2.27 Reset-node PDUs

Rest-node-req-PDU specific values are given in Table 64.

Table 64 – Rest-node-req-PDU specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FE04’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

Rest-node-rsp-PDU specific values are given in Table 65.

Table 65 – Rest-node-rsp-PDU specific values

Field	Value
TFL	‘40’H
PPT	‘1’B
RPL	Not used. ‘0’B
ULS	Not used. ‘0’H
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
MFT	Not used. ‘0’H
TCD	‘FECC’H
C_AD1	Not used. ‘0’H
C_SZ1	Not used. ‘0’H
C_AD2	Not used. ‘0’H
C_SZ2	Not used. ‘0’H
Min_VER	‘0’H
MAJ_VER	3
Token	‘1’B
CBN	‘1’H
TBN	‘1’H
BSIZE	‘0’H
LKS	Not used. ‘0’H
TW	Not used. ‘0’H
RCT	Not used. ‘0’H

5.2.28 Cyclic-data PDUs

5.2.28.1 PDU specific values

Cyclic-data-PDU specific values are given in Table 66.

Table 66 – Cyclic-data-PDU specific values

Field	Value
SEQ	Not used. ‘0’H
RPL	with ACK, then ‘1’B
M_SZ	Not used. ‘0’H
M_ADD	Not used. ‘0’H
M_RLT	Not used. ‘0’H
TCD	‘FDE9’H

5.2.28.2 CyclicData

The CyclicData field contains the cyclic-data.

When the cyclic-data is greater than 1 024 octets, the cyclic-data is fragmented into two or more frames. The first frame contains the cyclic-data on the common-memory-area-1 and/or the common-memory-area-2, and the subsequent frames contain the cyclic-data on the common-memory-area-2.

When a node has ACKdata, the ACKdata is assembled into between the cyclic-data FALARHeader field and the CyclicData field.

The CyclicData shall be encoded as little endians.

5.2.28.3 ACKdata

The ACKdata field contains ACK as delivery confirmation sent out by the node which receives a confirm type message-data frame, and the ACK is conveyed to the sender node of the confirm type message-data frame by means of cyclic-data frame by the receiving node.

The ACKdata contains one or more ACKs which corresponds to the confirm type message-data frames received until the receiving node obtains the token and/or the receiving node sends out the cyclic-data frame with these ACKs.

A collection of the ACKs which is up to eight by a cyclic-data transmission is added between the cyclic-data FALARHeader field and the CyclicData field.

When the CyclicData is fragmented for the cyclic-data transmission, the ACKdata is assembled into the last cyclic-data frame or the final-cyclic-data frame.

The structure of the collection of eight ACKs is shown in Figure 51, and the value of the elements of ACKdata is shown in Table 67. Table 68 shows the value of R_STSx field. Data shall be encoded as big endians.

Octet-offset	Bit 0 Bit 7	Bit 8 Bit 15	Bit 16 Bit 23	Bit 24 Bit 31
0	A_VER	A_NUM	Reserved	
4	Reserved	R_STS1	R_TCD1	
8	‘0’H	‘1’H	‘0’H	R_NA1
12	R_VSEQ1			
16	R_SEQ1			
20	Reserved	R_STS2	R_TCD2	
24	‘0’H	‘1’H	‘0’H	R_NA2
28	R_VSEQ2			
32	R_SEQ2			
36	Reserved	R_STS3	R_TCD3	
40	‘0’H	‘1’H	‘0’H	R_NA3
44	R_VSEQ3			
48	R_SEQ3			
⋮	⋮			
116	Reserved	R_STS8	R_TCD8	
120	‘0’H	‘1’H	‘0’H	R_NA8
124	R_VSEQ8			
128	R_SEQ8			

Figure 51 – Structure of ACKdata

Table 67 – Value of the element of ACKdata

Data element	Value	Description
A_VER	0	Fixed number for Version of ACKdata
A_NUM	1..8	Total number of ACKs
R_STSx	1..6	Status identifier. See Table 68.
R_TCDx	10 000..65 535	TCD number received corresponding to this ACKdata
R_NAx	1..254	Destination node number to deliver this ACKdata
R_VSEQx	'1'H..'FFFF FFFF'H	Version of sequence number received
R_SEQx	'1'H..'FFFF FFFF'H	Sequence number received
Reserved	'0'H	Reserved for future extension

Table 68 – Value of R_STSx field

Value	Description
01	Successfully received without error
02	Receive buffer full
03	Message reception process is not yet completed
05	Version of sequence number error
06	Invalid format error

6 FAL protocol state machines structure

6.1 Overview

FAL protocol state machine is composed of three kinds of integrated protocol machines: the FAL service protocol machine (FSPM), the application relationship protocol machine (ARPM) and the data-link layer mapping protocol machine (DMPM).

Figure 52 shows the architectural relationships among these protocol machines.

The FSPM provides an interface to FAL service users and performs basically the mapping between the FAL-user services and the FAL internal services. Service primitives from the FAL service users are converted into internal primitives, and the internal primitives are passed to appropriately selected protocol machines, and internal primitives received are converted and forwarded as confirmations and responses to the FAL service users.

The ARPM converts service primitives and forwards them between the ARPM and the DMPM. The ARMP is responsible for FAL-user services related procedures coordination; the token passing control to establish and maintain a logical ring for right to transmit, node addition, node drop-out and the logical ring recovery; the data transmission and coordination for forwarding APDUs to the data-link layer to send out as data transmission frames on the medium, the error handling with delivery confirmation and retransmission.

The DMPM specifies the mapping into the data link layer.

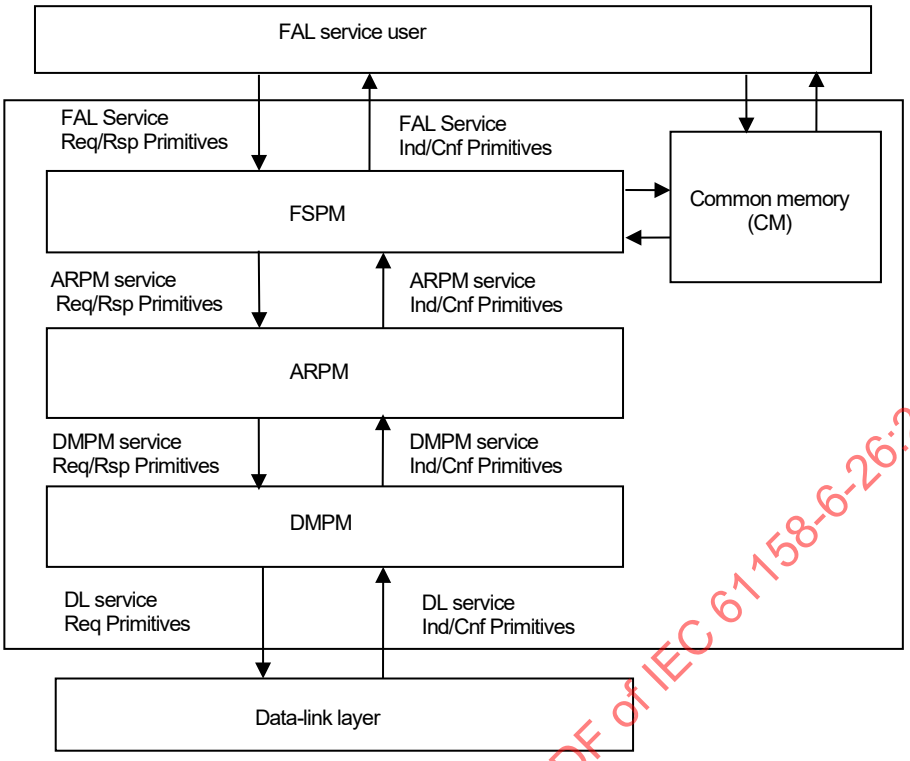


Figure 52 – Relationship between FAL protocol machines

6.2 Common variables, parameters, timers, counters, lists and queues

6.2.1 V(3CWT), P(3CWT), T(3CWT): Three-lap-time-period-of-the-token-circulation

This variable holds and designates the three-lap-time-period-of-the-token-circulation that is the three-lap time period of the token circulation. A new node in the in-ring start-up state waits for the T(3CWT) time out and starts the T(PWT) to send out a participation-request frame for entering into a logical ring in operation, and the T(3CWT) is used for the retransmission control for entering into a logical ring and for message-data transmission after sending out a message-data frame.

The value shall be set to 3 000 ms.

6.2.2 V(ACK): ACK received

This variable indicates whether the ACK data from other nodes contains the ACK to this node.

The value “True” indicates the ACK to this node exists and “False” indicates no ACK.

6.2.3 V(ACK_TN): ACK to this node

This variable holds the ACK destined for this node sent by other nodes with the final-cyclic-data frame and contains the elements as follows:

– V(ACK_TN).STS: Reception status at other nodes: refer to Table 69,
– V(ACK_TN).TCD: Corresponding TCD number received at other nodes,
– V(ACK_TN).VSEQ: Version of sequence number received at other nodes,
– V(ACK_TN).SEQ: Sequence number received at other nodes.

Table 69 – Value of R_STSx field

Value	Description
01	Successfully received without error
02	Receive buffer full
03	Message reception process is not yet completed
05	Version of sequence number error
06	Invalid format error

6.2.4 V(AWT), P(AWT), T(AWT): Waiting-time-period-for-receiving-message-acknowledge

This variable holds and designates the waiting-time-period-for-receiving-message-acknowledge that is the time period for a node to wait an ACK from other nodes after sending out a message-data frame to other nodes.

T(AWT) is used to wait the time period.

The value shall be set to 100 ms.

6.2.5 V(CBN): Current fragment number for fragmented cyclic-data transmission

This variable holds the current fragment number of cyclic-data transmission.

6.2.6 V(CTFG): Cyclic-data fragment transfer

This variable indicates fragmented cyclic-data exists to be sent out.

The value “True” indicates fragmented cyclic-data remained and “False” indicates no more fragmented cyclic-data.

6.2.7 V(CTRen), P (CTRen): Cyclic-data receive enable

This variable holds and designates the Cyclic-data reception enable as a subscriber.

The value “True” indicates this node is a subscriber of Cyclic-data transmission and “False” indicates no subscriber.

6.2.8 V(CTRQ): Cyclic-data transfer request

This variable indicates whether any cyclic-data exists to be sent out.

The value “True” indicates the cyclic-data exists and “False” indicates no cyclic-data.

6.2.9 C(MCNT): Cumulative count of message transmission carried over

C(WCNT) indicates the number of cumulative count of message transmission carried over when a node obtains the token and has message-data to send out.

6.2.10 V(MCV): Message transmission carried over

This variable indicates whether a message-data transmission carried over or not during this time period a node obtains the token.

The value “True” indicates the message-data transmission is attempted but is carried over in this time a node obtains the token.

6.2.11 V(NMTP): No message transmission in previous cycle

This variable holds and indicates whether a node sent out any message-data at the previous time the node obtained the token.

The value “True” indicates the node sent out no message-data and “Fault” indicates the node sent out any message-data in the previous token-circulation time.

6.2.12 V(MFT), P(MFT), T(MFT): Allowable-minimum-frame-Interval-Time

This variable holds and designates the minimum-frame-interval-time-period of the node, which is the time-period from the end of a frame received or sent to the subsequent frame to be sent.

V(MFT) applies:

a) Time period from the time a node obtains the token until the node starts to send out any frame;

b) Time period from the time a node completes to send out any frame until the time the node starts to send out any subsequent frame. V(MFT) shall not apply to the time period between the end of sending out the final-cyclic-data frame and the start of sending out the token frame. The time period shall be at the least time of each node.

The V(MFT) values of other nodes is shared by means of checking the APDU header received from other nodes, and the node can refer to the maximum value and can use the value as the V(MFT).

T(MFT) is used to monitor the time period.

The range of the value shall be 0 to 50. The unit is 100 μs.

6.2.13 V(MmtCntType): Measurement control type

This variable is used to specify and indicate the control type for the communication load measurement.

The values of this variable are as follows:

– 0x1: Activate Token-holding-time measurement;
– 0x81: Deactivate Token-holding-time measurement;
– 0x2: Activate GP_Comm sender log measurement;
– 0x82: Deactivate GP_Comm sender log measurement.

6.2.14 V(MRVRQ): Message receive request

This variable indicates whether any message-data received exists to be processed.

The value “True” indicates message-data exists and “False” indicates no message-data.

6.2.15 V(MSRQ): Message transfer request

This variable indicates whether any message-data exists to be sent out.

The value “True” indicates message-data exists and “False” indicates no message-data.

6.2.16 Q(MSRXQ): Message-RX-Queue

This Queue is used to queue into and to hand over the message data of other nodes received by the Message-data transmission on a FIFO basis.

6.2.17 Q(MTXQ):Message-TX-Queue

This Queue is used to queue into and to hand over the ASDU for message-data transmission on a FIFO basis.

6.2.18 V(PAT), P(PAT), T(PAT): Participation-request-frame-acceptance-time

This variable holds and designates the participation-request-frame-acceptance-time that is the time period for a node to wait the reception of any participation-request frames from other nodes after the node receives any trigger frames from other nodes or the node sends out a trigger frame in the network start-up state.

T(PAT) is used to wait the time period.

The value shall be set to 1 200 ms.

6.2.19 V(PnMgtIF): Participation-node-management-information List

Participation-node-management-information List is a structured collection of the Participation-node-management-information-parameters of each node, each of which contains the following parameter values: Node number, ULS, C_AD1, C_SZ1, C_AD2, C_SZ2, RCT, TW, MFT, VDN, MSN, NDN, P_Type, LKS, Node_status and node class. Refer to 5.2.1 and 5.2.4. This Participation-node-management-information List includes the Participation-node-management-information-parameters of this node.

6.2.20 V(PWT), T(PWT): Participation-request-frame-transmission-waiting-time

This variable holds the participation-request-frame-transmission-waiting-time that is the time period for a node waiting to send out a participation-request frame after receiving any trigger frames or sending out a trigger frame in the network start-up state of the node.

T(PWT) is used to wait the time period.

V(PWT) of a node is calculated based on the node number “N”. That is to circumvent overlapping each other the timing for new nodes to start sending out a participation-request frame.

The value is (N x 4) ms.

6.2.21 V(RCT): Allowable-refresh-cycle-time

This variable holds the allowable-refresh-cycle-time that is the 120 % value of the time period within which a node obtains the token again after sending out a token frame under the condition of no reception of any message-data frames in normal, steady state.

V(RCT) is determined dynamically after receiving the token three times in normal, steady state. In the network start-up state and in the node addition state of a node, V(RCT) shall be set to zero until after receiving the token three times, and any message-data frame shall not be sent out of the node.

The range of the value shall be 0 to 65 535, and the unit is 1 ms.

6.2.22 V(RMT), T(RMT): Refresh-cycle-measurement-time

This variable holds the refresh-cycle-measurement-time that is the time period within which a node obtains the token again after sending out a token frame in normal, steady state.

T(RMT) is used to measure the time period.

The range of the value shall be 0 to 65 535, and the unit is 1 ms.

6.2.23 C(RTX): Retransmission count

This variable holds the cumulative count of message-data retransmission. The maximum counts up to three are permitted for message-data retransmission.

6.2.24 V(SEQ): Sequence number value List

Sequence number value List is a structured collection of the SEQ values of each node, each of which indicates the latest SEQ value received from other nodes. This Sequence number value List includes the SEQ value of this node that is the latest SEQ number the node used at the time of message-data transmission.

6.2.25 V(SN): Successor node

This variable holds the node number of which a node is the next node for the token to be passed from the token holding node at that time. The possible value is 0 to 254, and the value 0 (zero) represents no other participating nodes than this node.

6.2.26 V(SrtMmt): Measurement started

This variable indicates the communication load measurement is in progress or suspended.

The value “True” indicates the measurement in progress, and the value “False” indicates the measurement suspended.

6.2.27 Q(SVRXQ): Server-RX Queue

This Queue is used to queue into and to hand over the server command message received on the connection by UDP protocol or TCP protocol or both of them on a FIFO basis.

6.2.28 Q(SVTXQ): Server-TX Queue

This Queue is used to queue into and to hand over the ASDU for the server command message transmission on the connection by UDP protocol or TCP protocol or both of them on a FIFO basis.

6.2.29 V(TBN), P(TBN): Total fragment number of Cyclic-data

This variable holds and designates the total fragment number of cyclic-data for the cyclic-data transmission.

6.2.30 V(TDT), P(TDT), T(TDT): Joining-token-detection-time

This variable holds and designates the joining-token-detection-time that is the time period for a node to monitor the activity of the token circulation over the fieldbus network in the entering state as a new node.

T(TDT) is used to monitor the time period.

Until after receiving any token frames and/or any trigger frames or after the T(TDT) time out, a node shall not send out any frames.

The value shall be set to 3 000 ms.

6.2.31 V(THT), P(THT), T(THT): Token-holding-time

This variable holds and designates the maximum time period within which a node shall release the token and send out a token frame. In the case that a node takes longer time than the V(TW), a token frame shall not be sent out in order to prevent multiple token circulations.

T(THT) is used to monitor the time period. V(THT) value shall not exceed V(TW).

The range of the value shall be 1 to 255, and the unit is 1 ms.

6.2.32 V(TK): Token holding

This variable indicates whether a node holds the token.

The value “True” indicates this node holds the token and “False” indicates no Token holding.

6.2.33 V(TKH): Token holding node

This variable indicates the node number of a token holding node at that time. The value is 0 to 254. The Value “0” indicates no other nodes than this node or no certain node settled.

6.2.34 V(TN): Node identifier number

This variable holds and designates the node identifier number of this node. The initial value is zero (0) on power-up or reset of this node.

The range of the value shall be 1 to 254.

6.2.35 V(TrWT), T(TrWT): Trigger-frame-transmission-waiting-time

This variable holds the trigger-frame-transmission-waiting-time that is the time period for a node waiting to send out a trigger frame after the T(TDT) time out.

The value of V(TrWT) is (R x 4) ms. R is the remainder of the node number divided by 8.

6.2.36 V(TSZ), P(TSZ): Total cyclic-data size

This variable holds and designates the total cyclic-data size assigned to this node for the Cyclic-data transmission.

6.2.37 V(TW), P(TW), T(TW)( ): Token-watchdog-time

This variable holds and designates the maximum observational time period of a node for the token circulation, which is the elapsed time period from the time the node obtains the token until the time the node sends out a token frame over the fieldbus network. The V(TW) values of other nodes are shared by means of checking the APDU header received from other nodes, and the node can utilize these V(TW).node-number to detect some kind of failure by means of checking the elapsed time period during which a node holds the token.

T(TW)( ) is used to monitor the time period for the token circulation.

The range of the value shall be 1 to 255 and the unit is 1 ms.

6.2.38 V(VSEQ): Version of sequence number value List

Version of sequence number values List is a structured collection of the VSEQ values of each node, each of which indicates the latest VSEQ value received from other nodes. This Version of sequence number values List includes the VSEC value of this node.

6.3 Functions used in state tables

The functions used in each protocol machines are summarized in Table 70.

Table 70 – Functions used in state tables

Function name	Input	Output	Operation
INIT_PROC	–	–	Initialize the node. All V( ) is set to the initial value of the P ( ) and the necessary process to initialize the node is performed.
SET_PROC	–	Status	Set up all the data-transmission related parameters; i.e. network configuration parameters, network management parameters and so on. The “Status” indicates whether the set-up process has completed or not. “True” indicates the completion, otherwise “False” is returned.
CLEAR_CMmt	Mmt-id	–	Clear all related counters to the Token-holding-time-measurement or the General-purpose-communication-sender-log measurement. “Mmnt-id” specifies: If “Mmnt-id” is “TKHmmt”, all counters related to the Token-holding-time-measurement are cleared; If “Mmnt-id” is “GPMmt”, all counters related to the General-purpose-communication-sender-log measurement are cleared.
BUILD_PDU	TCD, Data	APDU	According to requested TCD, assemble the corresponding APDU. If Data is specified, the APDU is assembled to append the Data after the APDU header fields.
UPDATE_Partici_node_mgt_info	Node number	–	According to requested Node number, update the corresponding parameters in the Participation-node-management-information.
CHECK_CM_ad	Node number, C_AD1, C_SZ1, C_AD2, C_SZ2	True/False	Check the requested Common-memory-address space is not overlapped. If no overlap, “True” is returned. Otherwise “False” is returned.
CHECK_NN	Node number	True/False	Check the requested Node number is not overlapped. If no overlap, “True” is returned. Otherwise “False” is returned.
Find_Smallest_NN	–	Node number	Find the smallest node number within the participating nodes. Value of 0 to 254 is returned. Value “0” denotes no other nodes participating.
Find_SN	Node number	Node number	Find the next node number for a node to hand over the token. Value of 0 to 254 is returned. Value “0” indicates no other nodes to hand over the token.
PUT_RX-BUFFER	T_sdu, P-id	Status	Store T_sdu into the receive-queue buffer associated with “P-id”. If the value of the P-id is “CS”, the receive-buffer is the Server-RX Queue. If the value of the P-id is “MT”, the receive-buffer is the Message-RX Queue. “Status” indicates the buffer condition; “Full” is of the Buffer full, “Not full” is not full and “Empty” is empty.
GET_RX-BUFFER	P-id	T_sdu	Get T_sdu in the receive-queue buffer associated with “P-id”. If the value of the P-id is “CS”, the receive-buffer is the Server-RX Queue. If the value of the P-id is “MT”, the receive-buffer is the Message-RX Queue.

Function name	Input	Output	Operation
GET_TX-BUFFER	P-id	ASDU	Get ASDU in the TX-queue buffer. ASDU consists of the corresponding APDU and Channel-type. Channel-type indicates UDP. If the value of the P-id is “CS”, the TX-queue buffer is the Server-TX Queue. If the value of the P-id is “MT”, the TX-queue buffer is the Message-TX queue.
PUT_TX-BUFFER	ASDU, P-id	Status	Store ASDU into the TX-queue buffer. ASDU consists of the corresponding APDU and Channel-type. Channel-type indicates UDP or TCP channel. If the value of the P-id is “CS”, the TX-queue buffer is the Server-TX Queue. If the value of the P-id is “MT”, the TX-queue buffer is the Message-TX queue. “Status” indicates the buffer condition; “True” indicates the buffer is not empty and “False” indicates empty.
BUILD_ASDU	APDU, Channel-type	ASDU	Assemble a corresponding ASDU according to the APDU and channel-type. Channel-type specifies UDP or TCP channel type.
BUILD_TSDU	APDU, Channel-type, P-id	TSDU	Assemble a corresponding T_SDU according to the APDU and channel-type. Channel-type specifies UDP or TCP channel type. P-id specifies “CS” or “MT”. if P-id = “MT” then if TCD =Cyclic-data-PDU || Token-PDU then DSAP:= 55 000; SSAP:= 55 003 else if TCD = Participation-req-PDU || Trigger-PDU then DSAP:= 55 002; SSAP:= 55 003 else DSAP:= 55 001; SSAP:= 55 003 else DSAP:= 55 004; SSAP:= 55 004 endif
CHECK_ACK	–	True/False	Check a received ACK-data whether ACK for this node from other nodes exists. If ACK exists, “True” is returned, otherwise “False” is returned.
GET_ACK	–	ACK_TN	Get ACK data destined for this node out of the received ACK data in a final-cyclic-data frame received. The ACK_TN contains: – ACK_TN.STS: Reception status from other nodes, – ACK_TN.TCD: Corresponding TCD number, – ACK_TN.VSEQ: Version of sequence number, – ACK_TN.SEQ: Sequence number received.
BUILD_ACK	R_STS, R_TCD, R_NA, R_VSEQ, R_SEQ	ACK_data	Assemble an ACK data for ACK data delivery to other nodes.
BUILD_ACKHdr	A_VER, A_NUM	ACK_Hdr	Assemble an ACK data header for ACK data delivery to other nodes
QUEUE_ACK	Ack_data or Null	Count	Queues the input data into the ACK Queue on a FIFO basis. The “Count” shows the total count of the queued data.
DEQUEUE_ACK	Count	Ack_data	Dequeue up to “Count” queued data from the ACK Queue.
GET_BUFF_CT-TX	Offset, Size	Data	Get Cyclic-data on the Cyclic-TX-buffer from specified “Offset” address with the specified memory “Size”.
PUT_BUFF_CT-TX	Data, Size	–	Store the “Data” with the “Size” into the Cyclic-TX-buffer.

Function name	Input	Output	Operation
PUT_BUFF_CT-RX	Data, S-Offset, D-Offset, Size	–	Store the portion of “Data” which resides at the address specified by the “S-Offset” with length of “Size”, into the Cyclic-RX-buffer of the starting address specified by the “D-Offset” with the specified memory length of “Size”.
GET_BUFF_CT-RX	Offset, Size	Data	Get the Data which resides at the address specified by the “Offset” with length of “Size” on the Cyclic-RX-buffer.
Update_CM	Data, C_AD, C_SZ	–	Replace the memory area on the local Common memory, which is specified by the head-address of C_AD with the address size of C_SZ, with Data
GET_CT_OffSize	TBN, CBN, TSZ	Offset, Size	Calculate the offset address and the size with specified TBN and CBN number as follows: Offset = CBN x 1 024 if TBN > CBN+1 then Size = 1 024 else if TBN <= CBN +1 then Size = TSZ – CBN x 1 024 endif
SAVE_TSDU	T_sdu	–	Save the data into TSDU-buffer
RESTORE_TSDU	–	T_sdu	Restore the data from TSDU-buffer
BIND	Data1, Data2	Data	Compile “Data1” and “Data2” in this order as a “Data”.
CHECK_TSDU	T_sdu	True/False	Check the validity of the specified T_sdu. If valid, “True” is returned, otherwise “False” is returned.
CLEAR_LogArea	–	Status	Clear the Log area and the Status” is returned. The value of the Status is identical to M_RLT; i.e. 0x0: Success – Successfully completed; 0x1: Failure – Invalid requested parameter; 0x2: Failure – Service unimplemented.
CS_PROC	TCD, Parameter-list, Data	Status, Result data	Handle a server-command requested, and the result status is returned. The “Status” indicates as follows: – 0x0: Successfully completed without Data – 0x80: Successfully completed with Data – 0x1: Invalid requested parameter without Data – 0x81 Invalid requested parameter with Data – 0x2: Service unimplemented
CHECK_NParaChange	–	Status	Check whether the configuration parameters related to the node network parameters is changed. “Status” indicates the result. If changed, “True” is returned, otherwise “False” is returned.
INIT-TIME_DELAY	V(X)	–	Wait by the time period specified by v(x).
GET_CURRENT_VAL	VAL names/ID	Current values Status	Get the values of variables specified by the Variable-names or Identifier
SET_VALUE	VAL names/ID, Desired values	Status	Set the Desired-values to the Variables specified by the Variable-names or Identifier

7 FAL service protocol machine (FSPM)

7.1 Overview

The role of FSPM is to provide an interface to FAL service users and to perform basically the mapping between the FAL-user services and the FAL internal services.

The FSPM receives service primitives from FAL service users and converts them into internal primitives; selects a protocol machine of the ARPM and forwards the internal primitives to the ARPM protocol machine; receives internal primitives from the ARPM and converts them into the FAL service primitives, and forwards them to the FAL service users.

The FSPM is composed of five protocol machines of Cyclic-data, Message-data, Load measurement, General purpose command server, and Network management.

Figure 53 shows the overall structure of the FSPM, the protocol machines, the interfaces and the relationship between the protocol machines.

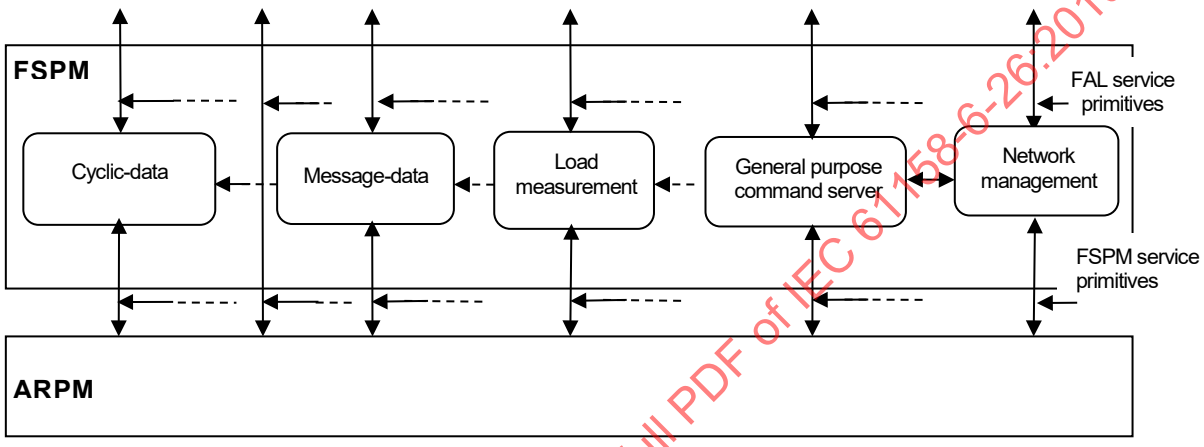


Figure 53 – Overall structure of FSPM

7.2 Cyclic-data protocol machine

7.2.1 Overview

The cyclic-data protocol machine provides the cyclic data communication interface to FAL service users and the mapping between the FAL-user services and the FAL internal services.

7.2.2 Cyclic-data primitives between FAL user and FSPM

The primitives between the FAL user and the FSPM are shown in Table 71.

Table 71 – Cyclic-data primitives between FAL user and FSPM

Primitive names	Source	Associated parameters	Description
Write.req	FAL user	C_AD1, C_AD2, C_SZ1, C_SZ2, Contents to write	Used to update the content of the CM
Write.cnf	FSPM	Result	Confirmation with the result
Send-CM.req	FAL user	C_AD1, C_AD2, C_SZ1, C_SZ2, Contents to write	Put the updated local CM data on the designated memory area to the send buffer and to send the data out of the buffer as a publisher
Send-CM.cnf	FSPM	Result	Confirmation with the result
Read.req	FAL user	C_AD1 and C_SZ1 or C_AD2 and C_SZ2,	Used to read the content out of the local Common memory
Read.cnf	FSPM	Contents read out, Result	Confirmation with the result
Update-memory.ind	FSPM	C_AD1, C_AD2, C_SZ1, C_SZ2	Inform the contents of the CM at the designated target memory are updated

Primitive names	Source	Associated parameters	Description
Get-buffer.cnf	FAL user	UpdateCM, UpdateAD1, UpdateAD2, C_AD1, C_AD2, C_SZ1, C_SZ2, Data	Used to get the data out of the receive buffer in which the updated CM data on the designated memory area contains, and to update the local CM data on the designated memory area as a subscriber

7.2.3 State table

The state transition diagram is shown in Figure 54, and the state table is shown in Table 72.

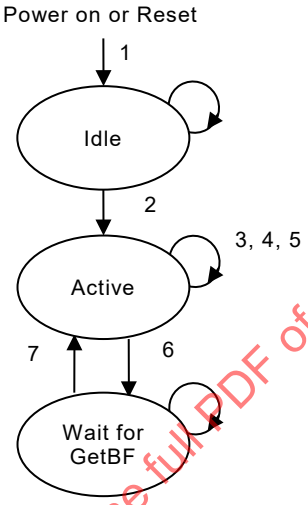


Figure 54 – State transition diagram of Cyclic-data protocol machine

Table 72 – State table of Cyclic-data protocol machine

#	Current state	Event / condition => actions	Next state
1	Any states	Power up or RESET => INIT_PROC ( ) INIT:= True	Idle
2	Idle	/ INIT = True => SetUp:= SET_PROC ( )	Active
3	ACTIVE	Write.req { C_AD1, C_AD2, C_SZ1, C_SZ2, Contents} => Update_CM (Contents, C_AD1, C_AD2, C_SZ1, C_SZ2) Write.cnf { “Completed without error”}	Active
4	ACTIVE	Send-CM.req { C_AD1, C_AD2, C_SZ1, C_SZ2, Contents} => Data:= Contents Size:= C_SZ1 + C_SZ2 if V(TBN) <> 0 && V(TSZ) <> 0 then PUT_BUFF_CT-TX (Data, Size) CT-send.req { } -- Request to publish updated Cyclic-data Send-CM.cnf { “Completed without error”} else Send-CM.cnf { “Invalid request – Not publisher” } endif	ACTIVE
5	ACTIVE	Read.req { C_AD, C_SZ } => Read.cnf { Data contents, “Completed without error”}	ACTIVE

#	Current state	Event / condition => actions	Next state
6	ACTIVE	ARUpdate-memory.ind { C_AD1, C_AD2, C_SZ1, C_SZ2} => UpdateCM:= True if (C_SZ1 <> 0 || C_SZ2 <>0) then if C_SZ1 <> 0 then UpdataeAD1:= True AD1:= C_AD1 SZ1:= C_SZ1 if C_SZ2 <> 0 then UpdateAD2:= True AD2:= C_AD2 SZ2:= C_SZ2 Update-memory.ind { UpdateAD1, C_AD1, C_SZ1, UpdateAD2, & C_AD2, C_SZ2}	Wait for GetBF
7	Wait for GetBF	Get-buffer.cnf /UpdateCM = True => if UpdateAD1 = True then Data:= GET_BUFF_CT-RX (0x0, C_SZ1) Update_CM (Data, C_AD1, C_SZ1) else if UpdateAD2 = True then Data:= GET_BUFF_CT-RX (C_SZ1, C_SZ2) Update_CM (Data, C_AD2, C_SZ2) endif UpdateCM:= False UpdateAD1:= False UpdateAD2:= False	ACTIVE

7.3 Message data protocol machine

7.3.1 Overview

Message data protocol machine provides the message-data communication function between nodes, and the message-data transmission is performed on demand by the FAL service user in order to intercommunicate between the APs running on remote nodes.

7.3.2 Message-data primitive between FAL user and FSPM

The primitives between the FAL user and the FSPM are shown in Table 73.

Table 73 – Message-data primitives between FAL user and FSPM

Primitive names	Source	Associated parameters	Description
Byte-block-read.req	FAL user	M_ADD, M_SZ, DNA	Used to read out the Byte-block-data of a target node
Byte-block-read.ind	FSPM	M_ADD, M_SZ, SNA	Indicate the Byte-block-read from other nodes
Byte-block-read.rsp	FAL user	SNA, Data read out, Result status	Response with data read out and the result status by the target node. SNA indicates the source node address from which the Byte-block-read-req is received.
Byte-block-read.cnf	FSPM	SNA, Data read out, Result status	Confirmation with data retrieved or error information as the result (-).SNA indicates the source node address from which the Byte-block-read-req is received.
Byte-block-read-rsp.cnf	FSPM	Result status	Confirmation with result (+/-) for sending out the Byte-block-read-rsp-PDU

Primitive names	Source	Associated parameters	Description
Byte-block-write.req	FAL user	M_ADD, M_SZ, DNA, Data	Used to write the Byte-block-data to a target node
Byte-block-write.ind	FSPM	SNA, M_ADD, M_SZ, Data	Indicate the Byte-block-read from other nodes
Byte-block-write.rsp	FAL user	SNA, M_ADD, M_SZ, Result status	Response with the result status
Byte-block-write.cnf	FSPM	SNA, M_ADD, M_SZ, Result status	Confirmation with data written or error information as result (-)
Byte-block-write-rsp.cnf	FSPM	Result status	Confirmation with result (+/-) for sending out the Byte-block-write-rsp-PDU
Word-block-read.req	FAL user	M_ADD, M_SZ, DNA	Used to read out the Word-block-data of a target node
Word-block-read.ind	FSPM	SNA, M_ADD, M_SZ	Indicate the Word-block-read from other nodes
Word-block-read.rsp	FAL user	SNA, M_ADD, M_SZ, Data required, Result status	Response with data required and the result status
Word-block-read.cnf	FSPM	SNA, Data retrieved, Result status	Confirmation with data retrieved or error information as the result (-)
Word-block-read-rsp.cnf	FSPM	Result status	Confirmation with result (+/-) for sending out the Word-block-read-rsp-PDU
Word-block-write.req	FAL user	M_ADD, M_SZ, DNA, Data	Used to write the Word-block-data to a target node
Word-block-write.ind	FSPM	M_ADD, M_SZ, SNA, Data	Indicate the Word-block-write from other nodes
Word-block-write.rsp	FAL user	SNA, Result status	Response with the result status
Word-block-write.cnf	FSPM	SNA, Result status	Confirmation with data written or error information with the result (-)
Word-block-write-rsp.cnf	FSPM	Result status	Confirmation with result (+/-) for sending out the Word-block-write-rsp-PDU
Network-parameter-read.req	FAL user	DNA	Used to retrieve the Network parameters of a target node
Network-parameter-read.ind	FSPM	SNA	Indicate the Network-parameter-read from other nodes
Network-parameter-read.rsp	FAL user	SNA, Data required, Result status	Response with data required and the result status
Network-parameter-read.cnf	FSPM	SNA, Data retrieved, Result status	Confirmation with data retrieved or error information as the result (-)
Network-parameter-read-rsp.cnf	FSPM	Result status	Confirmation with result (+/-) for sending out the Network-parameter-read-rsp-PDU
Network-parameter-write.req	FAL user	DNA, Data	Used to write the Network parameters to a target node
Network-parameter-write.ind	FSPM	SNA, Data	Indicate the Network-parameter-write from other nodes
Network-parameter-write.rsp	FAL user	SNA, Result status	Response with data written and the result status
Network-parameter-write.cnf	FSPM	SNA, Result status	Confirmation with data written or error information with the result (-)
Network-parameter-write-rsp.cnf	FSPM	Result status	Confirmation with result (+/-) for sending out the Network-parameter-write-rsp-PDU
Stop-command.req	FAL user	DNA	Used to set a target node to the STOP state
Stop-command.ind	FSPM	SNA	Indicate the Stop-command from other nodes
Stop-command.rsp	FAL user	SNA, Result status	Response with the result status

Primitive names	Source	Associated parameters	Description
Stop-command.cnf	FSPM	SNA, Result status	Confirmation with the result status
Stop-command-rsp.cnf	FSPM	Result status	Confirmation with result (+/-) for sending out the Stop-command-rsp-PDU
Operation-command.req	FAL user	DNA	Used to set a target node to the OPERATION state
Operation-command.ind	FSPM	SNA	Indicate the Operation-command from other nodes
Operation-command.rsp	FAL user	SNA, Result status	Response with the result status
Operation-command.cnf	FSPM	SNA, Result status	Confirmation with the result status
Operation-command-rsp.cnf	FSPM	Result status	Confirmation with result (+/-) for sending out the Operation-command-rsp-PDU
Profile-read.req	FAL user	DNA	Used to retrieve the Profile data of a target node
Profile-read.ind	FSPM	SNA	Indicate the Profile-read from other nodes
Profile-read.rsp	FAL user	SNA, Data required, Result status	Response with data required and the result status
Profile-read.cnf	FSPM	SNA, Data retrieved, Result status	Confirmation with data retrieved or error information as the result (-)
Profile-read-rsp.cnf	FSPM	Result status	Confirmation with result (+/-) for sending out the Profile-read-rsp-PDU
Transparent-msg.req	FAL user	Data, DNA, Transmission mode	Used to send out a user-defined-service message to other nodes. “Transmission mode” indicates whether the transmission mode is “P-to-P” or “Broadcast”.
Transparent-msg.ind	FSPM	Data, SNA, Transmission mode	Indicate the Transparent-msg from other nodes
Transparent-msg.cnf	FSPM	SNA, Result status	Confirmation with result status
Log-data-read.req	FAL user	DNA	Used to retrieve the Log data of a target node
Log-data-read.ind	FSPM	SNA	Indicate the Log-data-read from other nodes. This primitive is primarily used as an internal primitive within FSPM and actually replaced with Get-LogData.req to NMPM.
Log-data-read.rsp	FAL user	SNA, Data required, Result status	Response with data required and the result status
Log-data-read.cnf	FSPM	SNA, Data retrieved, Result status	Confirmation with data retrieved or error information as the result (-)
Log-data-clear.req	FAL user	DNA, Transmission mode	Used to clear the Log data of a target node
Log-data-clear.ind	FSPM	SNA, Transmission mode	Indicate the Log-data-clear from other nodes. This primitive is used primarily as an internal primitive within FSPM.
Log-data-clear.rsp	FAL user	SNA, Result status	Response with result status
Log-data-clear.cnf	FSPM	SNA, Result status	Confirmation with result status
Message-return.req	FAL user	Data, DNA	Used to conduct the message-data transmission test with a procedure that the target node sends back the received message data to the initiating node
Message-return.ind	FSPM	Data, SNA	Indicate the Message-return from other nodes. This primitive is primarily used as an internal primitive within FSPM.
Message-return.rsp	FAL user	SNA, Data received, Result status	Response with received data and the result status

Primitive names	Source	Associated parameters	Description
Message-return.cnf	FSPM	SNA, Data received, Result status	Confirmation with received data and the result status
Vendor-specific-msg.req	FAL user	DNA, VDN, SCODE, Data, Transmission mode	Used to request the vendor specific message to other nodes
Vendor-specific-msg.ind	FSPM	SNA, VDN, SCODE, Data, Transmission mode	Indicate the Vendor-specific-msg from other nodes
Vendor-specific-msg.rsp	FAL user	SNA, VDN, SCODE, Data, Result status	Response with data, VDN, SCODE and the result status
Vendor-specific-msg.cnf	FSPM	SNA, VDN, SCODE, Data, Result status	Response with data, VDN, SCODE and the result status
Vendor-specific-msg-rsp.cnf	FSPM	Result status	Confirmation with result (+/-) for sending out the Vendor-specific-msg-rsp-PDU
Set-remote-node-config-para.req	FAL user	Data, DNA	Used to set up the node configuration parameters to other nodes
Set-remote-node-config-para.ind	FSPM	Data, SNA	Indicate the Set-remote-node-config-para from other nodes
Set-remote-node-config-para.rsp	FAL user	SNA, Data set up, Result status	Response with data set up and the result status
Set-remote-node-config-para.cnf	FSPM	SNA, Data set up, Result status	Confirmation with data set up and the result status
Set-remote-node-config-para-rsp.cnf	FSPM	Result status	Confirmation with result (+/-) for sending out the Set-remote-node-config-para-rsp-PDU
Read-rmt-partici-node-mgt-info-para.req	FAL user	DNA, Node number	Used to read the participating node management information parameters of other nodes
Read-rmt-partici-node-mgt-info-para.ind	FSPM	SNA, Node number	Indicate the Read-rmt-partici-node-mgt-info-para from other nodes
Read-rmt-partici-node-mgt-info-para.rsp	FAL user	SNA, Data required, Result status	Response with data required and the result status
Read-rmt-partici-node-mgt-info-para.cnf	FSPM	SNA, Data retrieved, Result status	Confirmation with data retrieved or error information as the result (-)
Read-rmt-partici-node-mgt-info-para-rsp.cnf	FSPM	Result status	Confirmation with result (+/-) for sending out the Read-rmt-partici-node-mgt-info-para-rsp-PDU
Read-rmt- node-mgt-info-para.req	FAL user	DNA	Used to retrieve the node management information parameters of other nodes
Read-rmt- node-mgt-info-para.ind	FSPM	SNA	Indicate the Read-rmt- node-mgt-info-para from other nodes
Read-rmt- node-mgt-info-para.rsp	FAL user	SNA, Data required, Result status	Response with data required and the result status
Read-rmt- node-mgt-info-para.cnf	FSPM	SNA, Data retrieved, Result status	Confirmation with data retrieved or error information as the result (-)
Read-rmt- node-mgt-info-para-rsp.cnf	FSPM	Result status	Confirmation with result (+/-) for sending out the Read-rmt- node-mgt-info-para-rsp-PDU
Read-rmt-node-set-info-para.req	FAL user	DNA	Used to retrieve the node setting information parameters of other nodes
Read-rmt-node-set-info-para.ind	FSPM	SNA	Indicate the Read-rmt-node-set-info-para from other nodes
Read-rmt-node-set-info-para.rsp	FAL user	SNA, Data required, Result status	Response with data required and the result status
Read-rmt-node-set-info-para.cnf	FSPM	SNA, Data retrieved, Result status	Confirmation with data retrieved or error information as the result (-)

Primitive names	Source	Associated parameters	Description
Read-rmt-node-set-info-para-rsp.cnf	FSPM	Result status	Confirmation with result (+/-) for sending out the Read-rmt-node-set-info-para-rsp-PDU

7.3.3 State table

The state transition diagram is shown in Figure 55, and the state table is shown in Table 74.

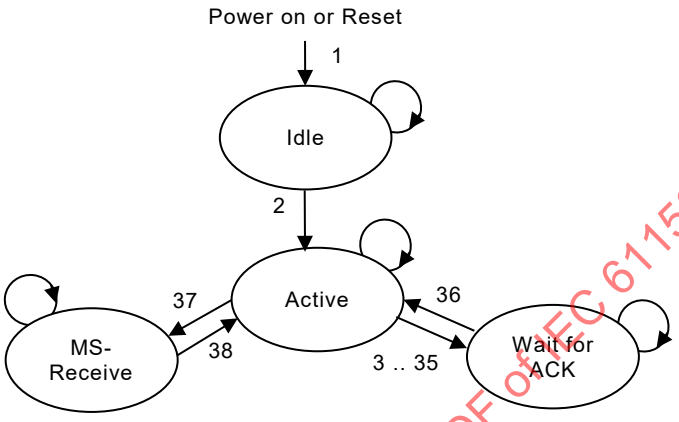


Figure 55 – State transition diagram of Message-data protocol machine

Table 74 – State table of Message-data protocol machine

#	Current state	Event / condition => actions	Next state
1	Any states	Power up or RESET => INIT_PROC ( ) INIT:= True	Idle
2	Idle	/ INIT = True => SetUp:= SET_PROC ( ) INIT:= False	Active
3	Active	Byte-block-read.req { M_ADD, M_SZ, DNA} => APDU:= BUILD_PDU (TCD = Byte-block-read-req-PDU, null) APDU.M_ADD:= M_ADD APDU.M_SZ:= M_SZ APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
4	Active	Byte-block-read.rsp { M_ADD, M_SZ, DNA, M_RLT, Data} => TCD = Byte-block-read-rsp-PDU APDU:= BUILD_PDU (TCD, Data) APDU.M_ADD:= M_ADD APDU.M_SZ:= M_SZ APDU.DNA:= DNA APDU.M_RLT:= M_RLT Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK

#	Current state	Event / condition => actions	Next state
5	Active	Byte-block-write.req { M_ADD, M_SZ, DNA, Data} => TCD = Byte-block-write-req-PDU APDU:= BUILD_PDU (TCD, Data) APDU.M_ADD:= M_ADD APDU.M_SZ:= M_SZ APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
6	Active	Byte-block-write.rsp { M_ADD, M_SZ, M_RLT, DNA } => TCD = Byte-block-write-rsp-PDU APDU:= BUILD_PDU (TCD, null) APDU.M_ADD:= M_ADD APDU.M_SZ:= M_SZ APDU.DNA:= DNA APDU.M_RLT:= M_RLT Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
7	Active	Word-block-read.req { M_ADD, M_SZ, DNA } => TCD = Worde-block-read-req-PDU APDU:= BUILD_PDU (TCD, null) APDU.M_ADD:= M_ADD APDU.M_SZ:= M_SZ APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
8	Active	Word-block-read.rsp { M_ADD, M_SZ, DNA, M_RLT, Data } => TCD = Word-block-read-rsp-PDU APDU:= BUILD_PDU (TCD, Data) APDU.M_ADD:= M_ADD APDU.M_SZ:= M_SZ APDU.DNA:= DNA APDU.M_RLT:= M_RLT Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
9	Active	Word-block-write.req { M_ADD, M_SZ, DNA, Data} => TCD = Word-block-write-req-PDU APDU:= BUILD_PDU (TCD, Data) APDU.M_ADD:= M_ADD APDU.M_SZ:= M_SZ APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK

#	Current state	Event / condition => actions	Next state
10	Active	Word -block-write.rsp { M_ADD, M_SZ, M_RLT, DNA } => TCD = Word -block-write-rsp-PDU APDU:= BUILD_PDU (TCD, null) APDU.M_ADD:= M_ADD APDU.M_SZ:= M_SZ APDU.DNA:= DNA APDU.M_RLT:= M_RLT Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
11	Active	Network-parameter-read.req {DNA } => TCD = Network-parameter-read-req-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
12	Active	Network-parameter-read.rsp { DNA, M_RLT, Data } => TCD = Network-parameter-read-rsp-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= DNA APDU.M_RLT:= M_RLT Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
13	Active	Network-parameter-write.req { DNA, Data } => TCD = Network-parameter-write-req-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
14	Active	Network-parameter-write.rsp { DNA, M_RLT } => TCD = Network-parameter-write-rsp-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA APDU.M_RLT:= M_RLT Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
15	Active	Stop-command.req { DNA } => TCD = Stop-command-req-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK

#	Current state	Event / condition => actions	Next state
16	Active	Stop-command.rsp { DNA, M_RLT } => TCD = Stop-command-rsp-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA APDU.M_RLT:= M_RLT Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
17	Active	Operation-command.req { DNA } => TCD = Operation-command-req-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
18	Active	Operation-command.rsp { DNA, M_RLT } => TCD = Operation-command-rsp-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA APDU.M_RLT:= M_RLT Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
19	Active	Profile-read.req { DNA } => TCD = Profile-read-req-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
20	Active	Profile-read.rsp { DNA, M_RLT, Data } => TCD = Profile-read-rsp-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= DNA APDU.M_RLT:= M_RLT Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
21	Active	Transparent-msg.req { DNA, Data, Trans-mode } => TCD = Transparent-msg-req-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= DNA APDU.M_Ctl (Bit 0, Bit 1):= Trans-mode Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
22	Active	Log-data-read.req { DNA } => TCD = Log-data-read-req-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK

#	Current state	Event / condition => actions	Next state
23	Active	Get-LogData.cnf { Data, M_RLT } => TCD = Log-data-read-rsp-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= V(LogRQNN) APDU.M_RLT:= M_RLT Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
24	Active	Log-data-clear.req { DNA, Trans-mode } => TCD = Log-data-clear-req-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA APDU.M_Ctl (Bit 0, Bit 1):= Trans-mode Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
25	Active	Message-return.req { DNA, Data } => TCD = Message-return-req-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
26	Active	Vendor-specific-msg.req { DNA, VDN, SCODE, Data, Trans-mode } => TCD = Vendor-specific-msg-req-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= DNA APDU.VDN:= VDN APDU.SCODE:= SCODE Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
27	Active	Vendor-specific-msg.rsp { DNA, VDN, SCODE, Data, M_RLT } => TCD = Vendor-specific-msg-rsp-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= DNA APDU.VDN:= VDN APDU.SCODE:= SCODE APDU.M_RLT:= M_RLT Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
28	Active	Set-remote-node-config-para.req { DNA, Data } => TCD = Set-remote-node-config-para-req-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK

#	Current state	Event / condition => actions	Next state
29	Active	Set-remote-node-config-para.rsp { DNA, Data } => TCD = Set-remote-node-config-para-rsp-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
30	Active	Read-rmt-partici-node-mgt-info-para.req { DNA, Node number } => TCD = Read-rmt-partici-node-mgt-info-para-req-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
31	Active	Read-rmt-partici-node-mgt-info-para.rsp { DNA, Data } => TCD = Read-rmt-partici-node-mgt-info-para-rsp-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
32	Active	Read-rmt- node-mgt-info-para.req { DNA } => TCD = Read-rmt- node-mgt-info-para-req-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
33	Active	Read-rmt- node-mgt-info-para.rsp { DNA, Data } => TCD = Read-rmt- node-mgt-info-para-rsp-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
34	Active	Read-rmt-node-set-info-para.req { DNA } => TCD = Read-rmt-node-set-info-para-req-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
35	Active	Read-rmt-node-set-info-para.rsp { DNA, Data } => TCD = Read-rmt-node-set-info-para-rsp-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK

#	Current state	Event / condition => actions	Next state
36	Wait for ACK	APDU.cnf { APDU, A_STS } => -- Result (+/–) if APDU.TCD = Byte-block-read-req-PDU then Byte-block-read.cnf { A_STS } else if APDU.TCD = Byte-block-read-rsp-PDU then Byte-block-read-rsp.cnf { A_STS } else if APDU.TCD = Byte-block-write-req-PDU then Byte-block-write.cnf { A_STS } else if APDU.TCD = Byte-block-write-rsp-PDU then Byte-block-write-rsp.cnf { A_STS } else if APDU.TCD = Worde-block-read-req-PDU then Word-block-read.cnf { A_STS } else if APDU.TCD = Word-block-read-rsp-PDU then Word-block-read-rsp.cnf { A_STS } else if APDU.TCD = Word-block-write-req-PDU then Word-block-write.cnf { A_STS } else if APDU.TCD = Word-block-write-rsp-PDU then Word-block-write-rsp.cnf { A_STS } else if APDU.TCD = Network-parameter-read-req-PDU then Network-parameter-read.cnf { A_STS } else if APDU.TCD = Network-parameter-read rsp-PDU then Network-parameter-read-rsp.cnf { A_STS } else if APDU.TCD = Network-parameter-write-req-PDU then Network-parameter-write.cnf { A_STS } else if APDU.TCD = Network-parameter-write-rsp-PDU then Network-parameter-write-rsp.cnf { A_STS } else if APDU.TCD = Stop-command-req-PDU then Stop-command.cnf { A_STS } else if APDU.TCD = Stop-command-rsp-PDU then Stop-command-rsp.cnf { A_STS } else if APDU.TCD = Operation-command-req-PDU then Operation-command.cnf { A_STS } else if APDU.TCD = Operation-command-rsp-PDU then Operation-command-rsp.cnf { A_STS } else if APDU.TCD = Profile-read-req-PDU then Profile-read.cnf { A_STS } else if APDU.TCD = Profile-read-rsp-PDU then Profile-read-rsp.cnf { A_STS } else if APDU.TCD = Transparent-msg-req-PDU then Transparent-msg.cnf {A_STS } else if APDU.TCD = Log-data-read-req-PDU then Log-data-read.cnf { A_STS } else if APDU.TCD = Log-data-read-rsp-PDU then NOP else if APDU.TCD = Log-data-clear-req-PDU then Log-data-clear.cnf { A_STS } else if APDU.TCD = Log-data-clear-rsp-PDU then NOP else if APDU.TCD = Message-return-req-PDU then Message-return.cnf { A_STS } else if APDU.TCD = Message-return-rsp-PDU then NOP else if APDU.TCD = Vendor-specific-msg-req-PDU then Vendor-specific-msg.cnf { A_STS } else if APDU.TCD = Vendor-specific-msg-rsp-PDU then Vendor-specific-msg-rsp.cnf { A_STS } else if APDU.TCD = Set-remote-node-config-para-req-PDU then Set-remote-node-config-para.cnf { A_STS } else if APDU.TCD = Set-remote-node-config-para-rsp-PDU then Set-remote-node-config-para-rsp.cnf { A_STS }	Active
36	Wait for ACK		Active

#	Current state	Event / condition => actions	Next state
		else if APDU.TCD = Read-rmt-partici-node-mgt-info-para-req-PDU then Read-rmt-partici-node-mgt-info-para.cnf { A_STS } else if APDU.TCD = Set-remote-node-config-para-rsp-PDU then Set-remote-node-config-para-rsp.cnf { A_STS } else if APDU.TCD = Read-rmt-partici-node-mgt-info-para-req-PDU then Read-rmt-partici-node-mgt-info-para.cnf { A_STS } else if APDU.TCD = Read-rmt-partici-node-mgt-info-para-rsp-PDU then Read-rmt-partici-node-mgt-info-para-rsp.cnf { A_STS } else if APDU.TCD = Read-rmt- node-mgt-info-para-req-PDU then Read-rmt- node-mgt-info-para.cnf { A_STS } else if APDU.TCD = Read-rmt- node-mgt-info-para-rsp-PDU then Read-rmt- node-mgt-info-para-rsp.cnf { A_STS } else if APDU.TCD = Read-rmt-node-set-info-para-req-PDU then Read-rmt-node-set-info-para.cnf { A_STS } else if APDU.TCD = Read-rmt-node-set-info-para-rsp-PDU then Read-rmt-node-set-info-para-rsp.cnf { A_STS } endif	
37	Active	MT_send.ind { RVT_sdu } => if !(PUT_RX-BUFFER (null, P-id = “MT”) = “Empty”) then RVT_sdu:= GET_RX-BUFFER (P-id = “MT”) V(MSRVRQ):= True endif	MS-Receive
38	MS-Receive	/ V(MSRVRQ) = True => if RVT_sdu.TCD = Byte-block-read-req-PDU then Byte-block-read.ind { M_ADD, M_SZ, DNA } else if RVT_sdu.TCD = Byte-block-read-rsp-PDU then Byte-block-read.cnf { M_ADD, M_SZ, DNA, M_RLT, Data } else if RVT_sdu.TCD = Byte-block-write-req-PDU then Byte-block-write.ind { M_ADD, M_SZ, DNA, M_RLT, Data } else if RVT_sdu.TCD = Byte-block-write-rsp-PDU then Byte-block-write.cnf { M_ADD, M_SZ, DNA, M_RLT } else if RVT_sdu.TCD = Worde-block-read-req-PDU then Worde-block-read.ind { M_ADD, M_SZ, DNA } else if RVT_sdu.TCD = Word-block-read-rsp-PDU then Word-block-read.cnf { M_ADD, M_SZ, DNA, M_RLT, Data } else if RVT_sdu.TCD = Word-block-write-req-PDU then Word-block-write.ind { M_ADD, M_SZ, DNA, M_RLT, Data } else if RVT_sdu.TCD = Word-block-write-rsp-PDU then Word-block-write.cnf { M_ADD, M_SZ, DNA, M_RLT } else if RVT_sdu.TCD = Network-parameter-read-req-PDU then Network-parameter-read.ind { DNA } else if RVT_sdu.TCD = Network-parameter-read-rsp-PDU then Network-parameter-read.cnf { DNA, M_RLT, Data } else if RVT_sdu.TCD = Network-parameter-write-req-PDU then Network-parameter-write.ind { DNA, Data } else if RVT_sdu.TCD = Network-parameter-write-rsp-PDU then Network-parameter-write.cnf { DNA, M_RLT } else if RVT_sdu.TCD = Stop-command-req-PDU then Stop-command.ind { DNA } else if RVT_sdu.TCD = Stop-command-rsp-PDU then Stop-command-rsp.cnf { DNA, M_RLT } else if RVT_sdu.TCD = Profile-read-req-PDU then Profile-read.ind { DNA } else if RVT_sdu.TCD = Profile-read-rsp-PDU then Profile-read.cnf { DNA, M_RLT, Data } else if RVT_sdu.TCD = Transparent-msg-req-PDU then Transparent-msg.ind {DNA, Data, Trans-mode } else if RVT_sdu.TCD = Log-data-read-req-PDU then V(LogRQNN):= RVT_sdu.SNA Get-LogData.req { RVT_sdu } -- To NMPM else if RVT_sdu.TCD = Log-data-read-rsp-PDU then Log-data-read.cnf {DNA, M_RLT, Data}	Active

#	Current state	Event / condition => actions	Next state
38	MS-Receive	else if RVT_sdu.TCD = Log-data-clear-req-PDU then Status:= CLEAR_LogArea ( ) TCD = Log-data-clear-rsp-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= RVT_sdu.SNA APDU.M_RLT:= Status APDU.M_Ctl.PPT:= True Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { } else if RVT_sdu.TCD = Log-data-clear-rsp-PDU then Log-data-clear.cnf {DNA, M_RLT, Data} else if RVT_sdu.TCD = Message-return-req-PDU then Data:= RVT_sdu.Data TCD = Message-return-rsp-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= RVT_sdu.SNA APDU.M_RLT:= 0x0 Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { } else if RVT_sdu.TCD = Vendor-specific-msg-req-PDU then Vendor-specific-msg.ind { DNA, VDN, SCODE, Data, Trans-mode } else if RVT_sdu.TCD = Vendor-specific-msg-rsp-PDU then Vendor-specific-msg.cnf { DNA, VDN, SCODE, M_RLT, Data} else if RVT_sdu.TCD = Set-remote-node-config-para-req-PDU then Set-remote-node-config-para.ind { DNA, Data } else if RVT_sdu.TCD = Set-remote-node-config-para-rsp-PDU then Set-remote-node-config-para.cnf { DNA, M_RLT, Data } else if RVT_sdu.TCD = Read-rmt-partici-node-mgt-info-para-req-PDU then Read-rmt-partici-node-mgt-info-para.ind { DNA, Node number } else if RVT_sdu.TCD = Read-rmt-partici-node-mgt-info-para-rsp-PDU then Read-rmt-partici-node-mgt-info-para.cnf { DNA, M_RLT, Data } else if RVT_sdu.TCD = Read-rmt- node-mgt-info-para-req-PDU then Read-rmt- node-mgt-info-para.ind { DNA } else if RVT_sdu.TCD = Read-rmt- node-mgt-info-para-rsp-PDU then Read-rmt- node-mgt-info-para.cnf { DNA, M_RLT, Data } else if RVT_sdu.TCD = Read-rmt-node-set-info-para-req-PDU then Read-rmt-node-set-info-para.ind { DNA } else if RVT_sdu.TCD = Read-rmt-node-set-info-para-rsp-PDU then Read-rmt-node-set-info-para.cnf { DNA, M_RLT, Data } endif V(MSRVRQ) = False	Active

7.4 Load measurement protocol machine

7.4.1 Overview

The load measurement protocol machine provides the function to conduct the start and the termination of the communication load measurement on the target node, and provides the measurement result containing the token holding time related and the general purpose communication load related statistical information on that node available for analyzing the total communication load condition over the Type 26 fieldbus network.

7.4.2 Load measurement primitives between FAL user and FSPM

The primitives between the FAL user and the FSPM are shown in Table 75.

Table 75 – Load measurement primitives between FAL user and FSPM

Primitive names	Source	Associated parameters	Description
Start-TK-hld-time-mrmt.req	FAL user	DNA	Used to request the start of the token-holding-time measurement to the target node
Start-TK-hld-time-mrmt.ind	FSPM	SNA	Indicate the Start-TK-hld-time-mrmt from other nodes. This primitive is primarily used as an internal primitive within the FAL, for instance for the statistics and can inform the FAL-user of the Token-holding-time measurement requested to start. The measurement start is activated on reception of Start-TK-hld-time-mrmt.req primitive to this node.
Start-TK-hld-time-mrmt.rsp	FAL user	SNA, Result status	Response with the result status
Start-TK-hld-time-mrmt.cnf	FSPM	SNA, Result status	Confirmation with the result status
Terminate-TK-hld-time-mrmt.req	FAL user	DNA	Used to request the termination of the token-holding-time measurement to the target node
Terminate-TK-hld-time-mrmt.ind	FSPM	SNA	Indicate the Terminate-TK-hld-time-mrmt from other nodes. This primitive is primarily used as an internal primitive within the FAL, for instance for the statistics and can inform the FAL-user of the Token-holding-time measurement requested to stop. The measurement stop is activated on reception of Terminate-TK-hld-time-mrmt.req primitive to this node.
Terminate-TK-hld-time-mrmt.rsp	FAL user	SNA, Result status, Measurement data	Response with the measurement data and result status
Terminate-TK-hld-time-mrmt.cnf	FSPM	SNA, Result status, Measurement data	Confirmation with the measurement data and result status
Start-GP_Comm-sndr-log.req	FAL user	DNA	Used to request the start of the General purpose-communication-sender log measurement to the target node
Start-GP_Comm-sndr-log.ind	FSPM	SNA	Indicate the Start-GP_Comm-sndr-log from other nodes. This primitive is primarily used as an internal primitive within the FAL, for instance for the statistics and can inform the FAL-user of the GP_Comm-sndr-log measurement requested to start. The measurement start is activated on reception of Start-GP_Comm-sndr-log.req primitive to this node.
Start-GP_Comm-sndr-log.rsp	FAL user	SNA, Result status	Response with the result status
Start-GP_Comm-sndr-log.cnf	FSPM	SNA, Result status	Confirmation with the result status
Terminate-GP_Comm-sndr-log.req	FAL user	DNA	Used to request the termination of the General purpose-communication-sender log measurement to the target node
Terminate-GP_Comm-sndr-log.ind	FSPM	SNA	Indicate the Terminate-GP_Comm-sndr-log from other nodes. This primitive is used to inform the FAL-user of the GP_Comm-sndr-log measurement requested to stop. The measurement stop is activated on reception of Terminate-GP_Comm-sndr-log.req primitive to this node.
Terminate-GP_Comm-sndr-log.rsp	FAL user	SNA, Result status, Measurement data	Response with the measurement data and the result status
Terminate-GP_Comm-sndr-log.cnf	FSPM	SNA, Result status, Measurement data	Confirmation with the measurement data and the result status

7.4.3 State table

The state transition diagram is shown in Figure 1, and the state table is shown in Table 76.

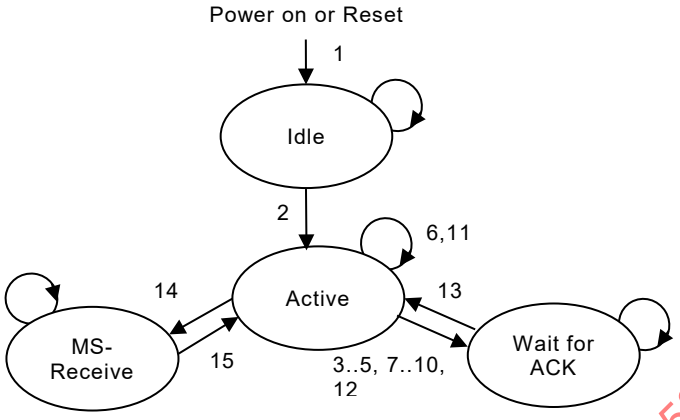


Figure 56 – State transition diagram of Load measurement protocol machine

Table 76 – State table of Load measurement protocol machine

#	Current state	Event / condition => actions	Next state
1	Any states	Power up or RESET => INIT_PROC ( ) INIT:= True	Idle
2	Idle	/ INIT = True => SetUp:= SET_PROC ( ) INIT:= False	Active
3	Active	Start-TK-hld-time-mrmt.req { DNA } => TCD = Start-TK-hld-time-mrmt-req-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
4	Active	Terminate-TK-hld-time-mrmt.req { DNA } => TCD = Terminate-TK-hld-time-mrmt-req-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK

#	Current state	Event / condition => actions	Next state
5	Active	Control measurement.cnf { Status } /(V(SrtMmt):= True) && (V(MmtRQNN) <> 0) && (V(MmtCntType) = 0x1) => -- Respond as TK-hld-time-mrmt in progress if Status = 0xFF then M_RLT:= 0x2 else if Status <> 0x1 then M_RLT:= 0x1 else M_RLT:= 0x0 endif DNA:= V(MmtRQNN) TCD = Start-TK-hld-time-mrmt-rsp-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA APDU.M_RLT:= M_RLT Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
6	Active	Control measurement.cnf { Status } / (V(MmtRQNN) <> 0) && (V(MmtCntType) = 0x81) => if Status = 0xFF then V(M_RLT):= 0x2 else if Status <> 0x81 then V(M_RLT):= 0x1 else V(M_RLT):= 0x0 endif V(LogRQNN):= V(MmtRQNN) Get-LogData.req { }	Active
7	Active	Get-LogData.cnf { Data, M_RLT } / (V(MmtRQNN) <> 0) && (V(MmtCntType) = 0x81) => -- Respond as TK-hld-time-mrmt suspended with measured result if !(M_RLT = 0x0) && (V(M_RLT) =0x0) then V(M_RLT) = M_RLT endif TCD = Terminate-TK-hld-time-mrmt-rsp-PDU APDU:= BUILD_PDU (TCD, Data) APDU.DNA:= V(LogRQNN) APDU.M_RLT:= V(M_RLT) Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { } V(SrtMmt):= False	Wait for ACK
8	Active	Start-GP_Comm-sndr-log.req { DNA } => TCD = Start-GP_Comm-sndr-log-req-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
9	Active	Terminate-GP_Comm-sndr-log.req { DNA } => TCD = Terminate-GP_Comm-sndr-log-req-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK

#	Current state	Event / condition => actions	Next state
10	Active	Control measurement.cnf { Status } /(V(SrtMmt):= True) && (V(MmtRQNN) <> 0) && (V(MmtCntType) = 0x2) => -- Respond as GP_Comm-sndr-log in progress if Status = 0xFF then M_RLT:= 0x2 else if Status <> 0x2 then M_RLT:= 0x1 else M_RLT:= 0x0 endif DNA:= V(MmtRQNN) TCD = Start-GP_Comm-sndr-log-rsp-PDU APDU:= BUILD_PDU (TCD, null) APDU.DNA:= DNA APDU.M_RLT:= M_RLT Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { }	Wait for ACK
11	Active	Control measurement.cnf { Status } / (V(MmtRQNN) <> 0) && (V(MmtCntType) = 0x82) => if Status = 0xFF then V(M_RLT):= 0x2 else if Status <> 0x82 then V(M_RLT):= 0x1 else V(M_RLT):= 0x0 endif V(LogRQNN):= V(MmtRQNN) Get-LogData.req { }	Active
12	Active	Get-LogData.cnf { Data, M_RLT } / (V(MmtRQNN) <> 0) && (V(MmtCntType) = 0x82) => -- Respond as GP_Comm-sndr-log suspended with measured result if !(M_RLT = 0x0) && (V(M_RLT) =0x0) then V(M_RLT) = M_RLT endif TCD = Terminate-GP_Comm-sndr-log-rsp-PDU APDU:= BUILD_PDU(TCD, Data) APDU.DNA:= V(LogRQNN) APDU.M_RLT:= V(M_RLT) Channel-type:= UDP ASDU:= BUILD_ASDU (APDU, Channel-type) PUT_TX-BUFFER (ASDU, “MT”) MT-send.req { } V(SrtMmt):= False	Wait for ACK
13	Wait for ACK	APDU.cnf { APDU, A_STS } => -- Result (+/–) if APDU.TCD = Start-TK-hld-time-mrmt-req-PDU then Start-TK-hld-time-mrmt.cnf { A_STS } else if APDU.TCD = Start-TK-hld-time-mrmt-rsp-PDU then NOP else if APDU.TCD = Terminate-TK-hld-time-mrmt-req-PDU then Terminate-TK-hld-time-mrmt.cnf { A_STS } else if APDU.TCD = Terminate-TK-hld-time-mrmt-rsp-PDU then NOP else if APDU.TCD = Start-GP_Comm-sndr-log-req-PDU then Start-GP_Comm-sndr-log.cnf { A_STS } else if APDU.TCD = Start-GP_Comm-sndr-log-rsp-PDU then NOP else if APDU.TCD = Terminate-GP_Comm-sndr-log-req-PDU then Terminate-GP_Comm-sndr-log.cnf { A_STS } else if APDU.TCD = Terminate-GP_Comm-sndr-log-rsp-PDU then NOP endif	Active
14	Active	MT_send.ind { T_sdu } / !(PUT_RX-BUFFER (Null, P-id = “MT”) = “Empty”) => RVT_sdu:= GET_RX-BUFFER (P-id = “MT”) V(MSRVRQ):= True	MS-Receive