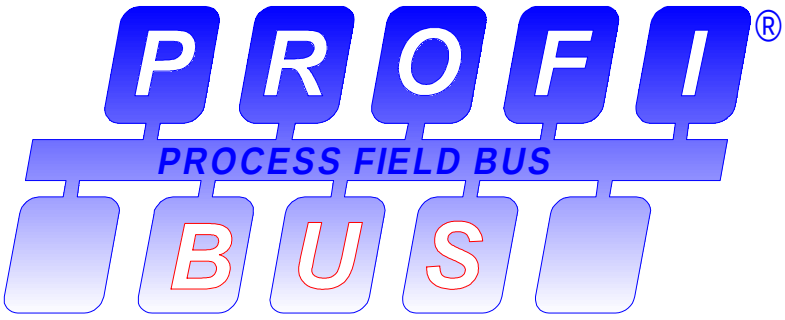
PROFIBUS



PROFIBUS

Specification

Normative Parts
of PROFIBUS -FMS, -DP, -PA
according to the European Standard
EN 50 170 Volume 2

Edition 1.0
March 1998

PROFIBUS Specification, Order No. 0.032

Foreword

PROFIBUS is a fieldbus system, which is in widespread use all over the world. PROFIBUS fulfils the requirements for the interconnection of intelligent field devices in manufacturing, process and building automation.

This publication contains the normative parts of the PROFIBUS-FMS / -DP / -PA specification according to the European fieldbus standard EN 50 170. These normative parts are supplemented by profiles and guidelines of the PROFIBUS user organisation (PNO). Both, the normative parts as well as the profiles and guidelines are published by PROFIBUS International, a world-wide organisation of more than 650 users of the PROFIBUS technology. PROFIBUS International is represented by regional user groups in 20 important industrial countries (see annex).

PROFIBUS has been selected by CENELEC TC65CX to be standardised as European Standard EN 50170 Volume 2 on 1996-07-02. Therefore, the PROFIBUS specification has been included without changes in EN 50170. This document is technical identical to the European Standard EN 50 170 volume 2 and is organised in the same manner.

PROFIBUS International
Business Office
Haid-und-Neu-Strasse 7
D-76131 Karlsruhe / Germany
Tel: ++ 49 721 9658 590
Fax: ++ 49 721 9658 589
email: PROFIBUS_International@compuserve.com
Internet: http://www.profibus.com

Table of Contents of the Normative Parts of the PROFIBUS Specification

This page is intentionally left blank.

PROFIBUS Specification - Normative Parts
Part 1
General Description of the Normative Parts

Contents

Page

1 General Description of the Normative Parts

This description provides an overview on the models, services and characteristics of the PROFIBUS System.

1.1 OSI Environment and layers defined

The layered structure is based on the ISO/OSI model for open systems communication (ISO 7498) according to figure 1. This specification defines Layer 1 (Physical Layer, PHY), Layer 2 (Data Link Layer, FDL) and Layer 7 (Application Layer). The Layers 3 to 6 are empty to minimize expense and increase efficiency.

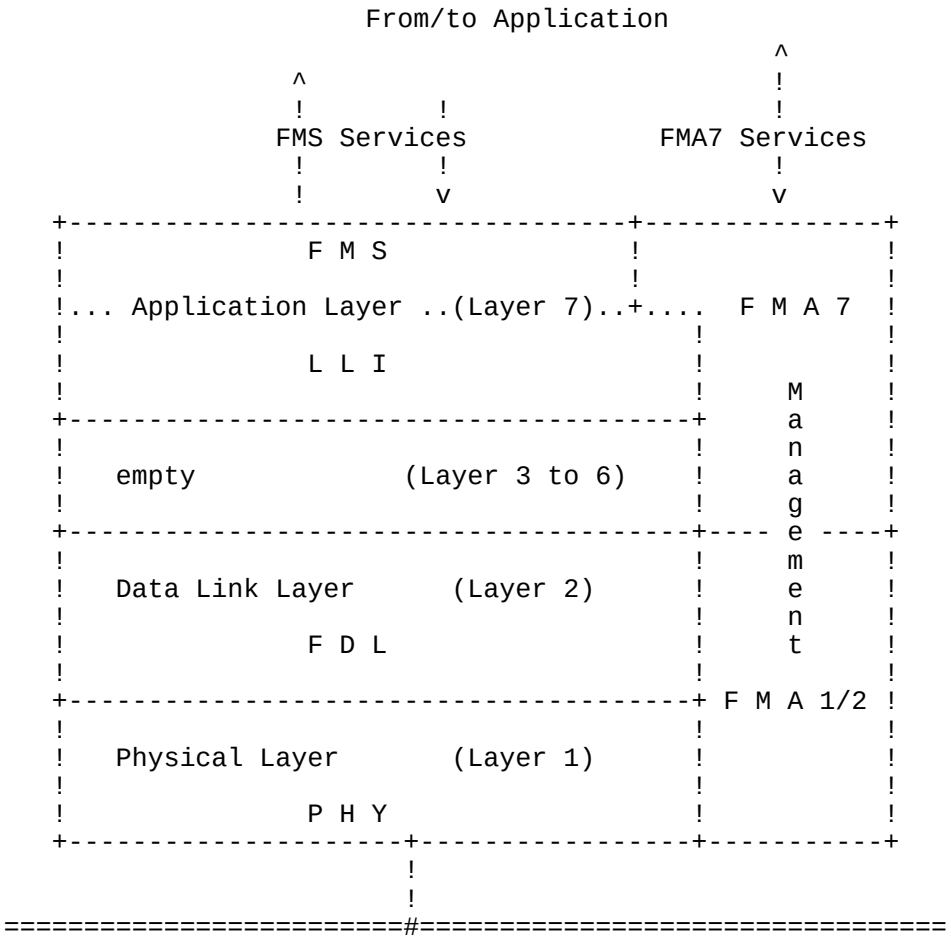


Figure 1. OSI environment

This specification defines the technical and functional characteristics of a serial fieldbus which is aimed at the inter-connection of digital field devices or systems with low or medium performance, e.g. sensors, actuators, transmitters, programmable logical controllers (PLC), numerical controllers (NC), programming devices, local man machine interfaces etc.

Often a field control system is based on a central control and supervision unit, which is connected to a number of devices and small systems distributed in the field. In such cases the dominant data transfer is centrally oriented and cyclic from the field devices to the central data processing unit or to a superior control system.

Strict realtime requirements have been met using the following simplifications:

- segmentation of long messages (> 235 byte) is not supported.
- blocking of short messages is not supported. The combination of many short messages into one long message packet is not in accordance with the requirement of short messages, and therefore the specification does not provide this function.
- support of routing functions by the network layer is not provided in the specification.
- except for a mandatory minimum configuration, arbitrary subsets of services can be created depending on the application requirements. This is a particularly important aspect for small systems (sensors, etc.).
- other functions are optional such as password protection schemes.
- the network topology is a linear bus with or without terminator, including drop cables and branches (tree)
- the medium, distances, number of stations Depending on the signal characteristics, e.g. for shielded twisted pair, ≤ 1,2 km without repeaters, 32 stations
- transmission speed depends on network topology and line lengths, e.g. stepwise from 9,6 to 1500 kbit/s
- a second medium (redundancy) is optional
- a halfduplex, asynchronous, slip protected synchronization (no bit stuffing) is used for transmission
- Data integrity of messages is Hamming distance (HD)=4, sync slip detection, special sequence to avoid loss and multiplication of data
- Addresses are defined in the range of 0 to 127 (127 = global addresses for broad-cast and multicast messages), address extension for regional address, segment address and Service Access address (Service Access Point, LSAP), 6 bit each
- two station types are used: (1) Masters (active stations, with bus access control), (2) Slaves (passive stations, without bus access control); preferably at most 32 masters, optionally up to 127, if the applications are not time critical.
- Bus access is based on a hybrid, decentral/central method; token passing between master stations and master-slave between master and slave stations. The token circulates in a logical ring formed by the masters. If the system contains only one master, e.g. a central control and supervision station, no token passing is necessary. This is a pure single-master/n-slave system. The minimum configuration comprises one master and one slave, or two masters.

- Data transfer services are

  (1) Acyclic:

  Send Data with/without Acknowledge
  Send and Request Data with Reply

  (2) Cyclic (Polling):

  Send and Request Data with Reply

1.2 Overview about the layers covered

The following describes the functionality of the various layers.

1.2.1 Physical Layer

The Physical Layer (this is the medium, including lengths and topology, the line interface, the number of stations and the transmission speed, variable in the range from 9,6 to 1500 kbit/s) can be adapted to different applications. However there is a common access method and transmission protocol and there are common services at the user interface.

The Physical Layer (version 1, contained in Part 2 of this specification) is specified according to the EIA standard RS-485:

- Topology : Linear bus, terminated at both ends, stubs ≤ 0,3 m, no branches;
- Medium : Shielded Twisted Pair
- Line Length : ≤ 1200 m
- Number of stations : 32 (master stations, slave stations or repeaters)
- Data rates : 9,6 / 19,2 / 93,75 / 187,5 / 500 / 1500 kbit/s

The version 2 (according to IEC 1158-2, contained in Part 9 of this specification) covers the requirements of Intrinsic Safety (IS):

- Topology : Linear bus, terminated at both ends, spurs ≤ 120 m
- Medium : Twisted Pair or multicore (shielded or unshielded)
- Line Length : ≤ 1900 m depending on cable type
- Number of stations : 32 (master stations, slave stations or repeaters)
- Data rates : 31,25 kbit/s

1.2.2 Data Link Layer

The Medium Access Control protocol (MAC), the data transfer services and the management services are defined according to the standards DIN 19 241-2, IEC 955(PROWAY C), ISO 8802-2 and ISO/IEC JTC 1/SC 6N 4960 (LLC type 1 and LLC type 3).

The octet (character) format shall be the UART format FT 1.2 (asynchronous transmission with start-stop synchronization) as defined for Telecontrol Equipment and Systems (IEC 870-5-1). The transmission protocol definitions are based on the standard IEC 870-5-2.

The following data transfer services are defined:

Send Data with Acknowledge (SDA)

This service allows an user to send data to a single remote station. If an error occurred, the data transfer shall be repeated.

Send Data with No Acknowledge (SDN)

This service allows an user to transfer data to a single remote station, to many remote stations (Multicast), or to all remote stations (Broadcast) at the same time without any confirmation.

Send and Request Data with Reply (SRD)

This service allows an user to transfer data to a single remote station and at the same time to request data from the remote station. If an error occured, the data transfer shall be repeated.

Cyclic Send and Request Data with Reply (CSRD)

This service allows an User to cyclically transfer data to a remote station and at the same time to request data from the remote station.

1.2.3 Application Layer

The Application Layer consists of the two entities FMS (Fieldbus Message Specification) and LLI (Lower Layer Interface).

1.2.3.1 Fieldbus Message Specification (FMS)

The FMS describes communication objects, services and associated models from the point of view of the communication partner (server behaviour).

The purpose of communication at the field and process levels is to transfer data (such reading/writing of measured values, loading/starting/stopping of programs, processing of events, etc.) between two communicating stations.

For communication between an application process of one device and that of another, the process objects transferred must be made known to the communication system, which means the process objects must be listed as communication objects in an Object Dictionary (OD) (comparable to a public telephone directory). Thus, an application process must make its objects visible and available, before these can be addressed and processed by the communication services. The application processes communicating with one another on different devices, however, need more information than the acknowledgement of the communication objects for efficient communication.

Usually, the stations are located at a distance from one another, or are not accessible during operation; so they must be uniquely identified with their features in the network. Data such as vendor name, model name and profile need to be read via the bus. Moreover, information on the communication interface status of the device and on the real device (e.g., indication of servicing dates) are a very important aspects.

A public Object Dictionary provided by all stations connected to the bus, standardized device features, identical services and uniform interfaces make up the basis for open communication between devices of different vendors. This consistent view of a device is called a Virtual Field Device (VFD).

The specification describes the effect of the services on the communication objects of an application process only for Virtual Field Devices. The mapping of Virtual Field Devices to real field devices and vice versa is not subject to the specification. Between the Application Layer and the real application process lies the so-called Application Layer Interface (ALI).

The services of the Application Layer can be accessed via this intermediate interface. It provides additional communication functions adapted to the application process. In addition, the Application Layer Interface does the mapping of the Virtual Field Device on to the real field device.

1.2.3.2 Lower Layer Interface (LLI)

Logical relationships exist between application processes with the specific purpose of transferring data. In the case, all communication relationships must be defined before a data transfer is started. These definitions are listed in layer†7 in the Communication Relationship List (CRL).

The manifold characteristics of the specification require a particular adaptation between FDL and FMS/FMA7. This adaptation is achieved by means of the LLI. The LLI is an entity of Layer 7.

The main tasks of LLI are:

- mapping of FMS and FMA7 services on to the FDL services
- connection establishment and release
- supervision of the connection
- flow control

1.2.4 User specifications

To manage peripheral devices connected through a serial interface to e. g. a controller in manufacturing application, this specification defines extented definitions to fulfil the special requirements in the area of remote peripherals.

Decentralized Peripherals (DP) are mainly used to connect automation systems (such as programmable controllers) via a fast serial link to input-/output-devices, sensors, actuators and to smart devices.

The main purpose of DP is the fast cyclic exchange of data between a powerful Master and several simple Slaves (peripheral devices). Thus, this system uses mainly the Master-Slave-type of communication services.

The hybrid media access allows Master-Slave communication as well as Master-Master communication, which is used for Data transfer between DP-Master (class 1) and DP-Master (class 2) for programmer/diagnostic-panels.

This page is intentionally left blank.

PROFIBUS Specification - Normative Parts

Part 2

Physical Layer Specification and Service Definition

Contents

1 Scope

This specification defines the functional, electrical and mechanical characteristics of a serial fieldbus which is designed for applications in field oriented automation systems.

An important requirement of these applications is the use of a simple transmission medium (2 or 4 wire cable) in a variety of topologies, such as the linear bus or tree topology. The support of low power and low cost implementations with real time behaviour (i.e. with guaranteed reaction time) is a further requirement. In many cases harsh electromagnetic interferences and sometimes hazardous areas, such as explosive atmospheres, have also to be taken into account.

In particular the specification supports simple input/output field devices without bus control capability as well as more sophisticated devices with bus control capability, e.g. Programmable Logical Controllers.

The requirements described above can be met by transmitting short messages efficiently and by realising the protocol in a commercially available integrated circuit, e.g. in a single chip microcontroller with an internal UART (Universal Asynchronous Receiver/Transmitter).

It is the purpose of this specification to minimise the costs of multi vendor field device interconnections, to integrate the components to a distributed control system and to guarantee reliable communication. Usually this is called "Open Systems Interconnection (OSI)". Further aims are the transparency and the easy access to higher level control systems.

The degree of freedom which is offerred by the specification guarantees flexibility and allows implementations for different applications, tailored to a variety of system configurations and functional structures.

This part of the specification defines the Physical Medium and the Physical Layer according to the ISO-OSI Layer Model (Ref. ISO 7498). The Data Link Layer, the interface between Data Link Layer and Application Layer, and the Management Layer are described in part 3 and 4.

The Physical Layer (version 1) is specified according to the EIA standard RS-485 and does not cover the requirements of Intrinsic Safety (IS). The version 2 (according to IEC 1158-2) covers the requirements of Intrinsic Safety (IS).

Part 5, 6 and 7 of this specification describes the Application Layer and the related management.

2 Normative References and additional Reference Material

DIN 19 241-2:1985	Messen, Steuern, Regeln; Bitserielles Prozessbus-Schnittstellensystem; Elemente des Übertragungsprotokolls und Nachrichtenstruktur
DIN 66 259-3:1983	Electrical characteristics for balanced double-current interchange circuits
DIN 57800/ VDE 0800:1984	Fernmeldetechnik; Errichtung und Betrieb der Anlagen
DIN VDE 0160:1988	Ausrüstung von Starkstromanlagen mit elektronischen Betriebsmittel

Source of Supply for DIN Standards:

Beuth Verlag GmbH
Burggrafenstr. 6

D-10772 Berlin
Germany

CCITT V.11:1976	Electrical Characteristics for balanced double-current interchange circuits for general use with integrated circuit equipment in the field of data communications.
CCITT V.24:1964	List of definitions for interchange circuits between data terminal equipment and data circuit-terminating equipment.
IEC 807-3:1990	Rectangular connectors for frequencies below 3 MHz Part 3: Detail specification for a range of connectors with trapezoidal shaped metal shells and round contacts Removable crimp contact types with closed crimp barrels, rear insertion/rear extraction
IEC 870-5-1:1990	Telecontrol equipment and systems; part 5: transmission protocols; section 1: transmission frame formats
IEC 870-5-2:1992	Telecontrol equipment and systems; part 5: transmission protocols; section 2: link transmission procedures
IEC 955:1989	Process data highway, type C (PROWAY C), for distributed process control systems
IEC 1131-2:1992	Programmable controllers - Part 2: Equipment requirements and tests
IEC 1158-2:1993	Fieldbus standard for use in industrial control system - Part 2: Physical layer specification and service definition
ISO 1177:1985	Information processing - Character structure for start/ stop and synchronous character oriented transmission

ISO 2022:1986	Information processing - ISO 7-bit and 8-bit coded character sets - Code extension techniques
ISO 7498:1984	Information processing systems; Open Systems Interconnection; Basic Reference Model
ISO 8802-2:1989	Information Processing Systems - Local area networks - Part 2: Logical link control
ISO/IEC JTC 1/ SC 6 N 4960:1988	Standards for Local Area Networks: Logical Link Control - Type 3 Operation, Acknowledged Connectionless Service
EIA RS-422-A:1978	Electrical Characteristics of balanced Voltage digital Interface Circuits
EIA RS-485:1983	Standard for electrical Characteristics of Generators and Receivers for use in balanced digital Multipoint Systems

3 General

3.1 Terms

The multiplicity of technical terms and the need to relate to existing international bus standards make it necessary to limit the technical terms to a well defined set.

3.1.1 Definitions

Definition	Meaning
Acknowledge Frame	conveys the status of a transaction from the remote (responding) FDL entity to the local (initiating) FDL entity
Action Frame	a data or request frame, the first frame transmitted in all "transactions"
Acyclic Service	an FDL Service that involves a single transfer or exchange of data, i.e. a single "transaction"
Address Extension	an LSAP address or Bus ID, it is conveyed at the start of the DATA_UNIT of a frame
BIT Time	FDL symbol period, the time to transmit one bit
Confirm(ation)	Confirm primitive e.g. informs the local FDL User of the success or failure and status of a transaction, also conveys Request Data, when present, to the local FDL User
Controller Type	generic implementation type (speed) and role of a station
current Master	the Master station that now holds the token (the token holder), the initiator of all transmissions
Cyclic Service	repetitions of an Acyclic FDL Service to multiple remote FDL Users according to a Poll List submitted by the local FDL User

Data Frame	an Action Frame that carries Send Data from a local FDL User to a remote FDL User
Data Request Frame	an Action Frame that carries Send Data and a request for Request Data from a local FDL User to a remote FDL User
Entity	the hardware/software embodiment of a protocol
FDL Status	the "token ring" status of a FDL entity conveyed in the FC field of reply frames
Fieldbus Management	protocol/entity that initializes, monitors, reports and supervises operation of FDL and PHY entities
Frame	the "packet" of data transmitted on the bus
Frame Header	the initial part of a frame that identifies its SA, DA and FC
GAP Maintenance	the actions taken (Request FDL Status) by this FDL entity to determine the FDL Status of all FDL entities in its GAP
Indication	Indication primitive e.g. informs the remote FDL User of the arrival of a Data or Request Frame and conveys Send Data, if present, to that remote FDL User
Live List	List of all live (respond to FDL Status requests) Master and Slave stations
local FDL User	the FDL User in the current Master station that initiates the current data transfer transaction
Poll Cycle	one execution of all the requests contained in the current Poll List
Poll List	the sequence of remote FDL Users to be addressed by the current Cyclic request
Request Data	Data provided by the remote FDL User to the local FDL User
remote FDL User	addressed (responding) FDL User of a "transaction"
Reply Frame	a Response or Acknowledge Frame, this frame completes a confirmed transaction
Request	Request primitive e.g. passes a request, to send Send Data and/or request Request Data or load Update, from the local FDL User to the local FDL entity
Request Frame	an Action Frame that carries a request for Request Data from a local FDL User to a remote FDL User
Response Frame	a Reply Frame that carries Request Data from a remote FDL User to a local FDL User
Send Data	data provided by a local FDL User to a remote FDL User
Token Holder	an attribute of the FDL entity in a Master station which confers the sole right to control (allocate) bus access to that station (the current Master)
Transaction	a complete confirmed or unconfirmed interaction (data transfer) between initiating (local) FDL or FMA1/2 Users and responding (remote) FDL Users or FDL or FMA1/2 entities

confirmed	Transaction consists of a single Action Frame and its responding Reply Frame along with a retry of these frames if no initial Reply Frame is received by the initiating Master station
unconfirmed	Transaction consists of a single Data Frame
Token pass	Transaction consists of the Token Frame, the confirmation of a successful pass and any retries needed unoccupied address the address of a non-existent station (does in GAPL not respond to Request FDL Status frames)

3.1.2 Abbreviations

A	Active Station
ACK	ACKnowledge(ment) frame
APP	APPlication layer (7), OSI layer 7
AWG	American Wire Gauge
Bus ID	Bus IDentification, an Address Extension (Region/Segment Address) that identifies a particular PROFIBUS segment as supporting routing between PROFIBUS segments (FDL)
CCITT	Comite Consultatif International Telephonique et Telegraphique
CNC	Computerized Numerical Control
CNTR	CoNTRol Signal
con	confirmation primitive
CSRD	Cyclic Send and Request Data with reply (FDL Service)
DA	Destination Address of a frame
DAE	Destination Address Extension(s) of a frame conveys DSAP and/or destination Bus ID
DGND	Data GrouND (PHY, RS-485)
DIN	Deutsches Institut fuer Normung e.V., Berlin German Industrial Standard & standards body
DMA	Direct Memory Access
DP RAM	Dual Port RAM (Random Access Memory)
DS	Disconnected Station local FDL/PHY controller not in logical token ring or disconnected from line (L/M_status of the service primitive)
DSAP	Destination Service Access Point a LSAP which identifies the remote FDL User
ED	End Delimiter of a frame
EIA	Electronic Industries Association
EMC	ElectroMagnetic Compatibility
EMI	ElectroMagnetic Interference
EXT	address EXTension field of a frame
FC	Frame Control (frame type) field of a frame

FCB	Frame Count Bit of a frame (FC field) used to eliminate lost or duplicated frames
FCS	Frame Check Sequence (checksum) of a frame used to detect corrupted frames
FCV	Frame Count bit Valid indicates whether the FCB is to be evaluated
FDL	Fieldbus Data Link layer, OSI layer 2
FMA1/2	Fieldbus MAnagement layer 1 and 2
FOW	Fiber Optic Waveguide
G	GAP update factor the number of token rounds between GAP maintenance (update) cycles
GAP	Range of station addresses from This Station (TS) to its successor (NS) in the logical token ring, excluding stations above HSA
GAPL	GAP List containing the status of all stations in this station's GAP
Hd	Hamming distance a measure of frame integrity, the minimum number of bit errors that can cause acceptance of a spurious frame
HSA	Highest Station Address installed(configured) on this PROFIBUS segment
IEC	International Electrotechnical Commission
IEEE	The Institute of Electrical and Electronics Engineers, USA
ind	indication primitive
ISO	International Organization for Standardization
L	Length of the information field the part of a frame that is checked by the FCS
LAS	List of Active (Master) Stations
LE	field giving LEngth of frame beyond fixed part
LEr	field that repeats LEngth to increase frame integrity
LR	Local Resource not available or not sufficient (L/M_status of the service primitive)
LS	Local Service not activated at service access point or local LSAP not activated (L/M_status of the service primitive)
LSAP	Link Service Access Point identifies one FDL User in a particular station
LSB	Least Significant Bit of a field or octet
L_pdu	Link_protocol_data_unit
L_pci	Link_protocol_control_information
L_sdu	Link_service_data_unit
LSS	Line Selector Switch (PHY), selects one of two redundant media
M	Master station, this station participates in the token ring; a full function station, its FDL users can initiate data transfer "transactions"

MC	Message Cycle
mp	Number of Retries per Poll List
MSAP	Management Service Access Point the MSAP used by this station's FMA1/2 entity for communication with other FMA1/2 entities
MSB	Most Significant Bit of a field or octet
mt	Number of Retries per token rotation
n	Number of Stations
na	Number of Active (Master) Stations
NA	No Acknowledgement/response
NIL	locally existing value, but not fixed in this standard
NO	Not OK (L/M_status of the service primitive)
np	Number of Slave (passive) Stations
NR	No Response FDL/FMA Data acknowledgement negative & send data ok (L/M_status of the service primitive)
NRZ	Non-Return-to-Zero (PHY), an encoding technique where transitions occur only when successive data bits have different values
NS	Next Station (FDL), the station to which this Master will pass the token
OK	Service finished according to the rules (L/M_status of the service primitive)
OSI	Open Systems Interconnection
PBC	PROFIBUS Controller
PC	Programmable Controller
PHY	PHYsical layer, OSI layer 1
PI	Parallel Interface
PON	Power ON transition occurs at a station
PS	Previous Station (FDL), the station which passes the token to this Master station
R	Receiver (PHY)
R_d	pulldown Resistance to DGND of bus terminator
RAM	Random Access Memory
RDH	Response FDL Data High and no resource for send data (L/M_status of the service primitive)
RDL	Response FDL/FMA1/2 Data Low and no resource for send data (L/M_status of the service primitive)
RET	RETry
REP	REPeater, a device that extends the distance and number of stations supported by a PROFIBUS System
req	request primitive
Res	Reserved

R_u	pullup Resistance to VP of bus terminator
RP	Reserved Power
RR	no Resource for send data and no Response FDL data (acknowledgement negative), (L/M_status of the service primitive)
RS	no Service or no Rem_add activated at Remote service access point(acknowledgement negative), (L/M_status of the service primitive)
RSAP	Reply Service Access Point an LSAP at which Request Data may be obtained
R_{SYS}	Rate SYStem total message cycles rate at which confirmed FDL Transactions are performed
R_t	line termination Resistance of bus terminator
RxD	Received Data signal (PHY, RS-485)
S	Slave station, this station can never hold the token, a reduced function station, its FDL User may only respond to requests from initiating FDL Users in Master stations
SA	Source Address of a frame
SAE	Source Address Extension(s) of a frame conveys SSAP and/or source Bus ID
SAP	Service Access Point, the point of interaction between entities in different protocol layers
SC	Single Character acknowledge frame
SD	Start Delimiter of a frame
SDA	Send Data with Acknowledge (FDL Service)
SDN	Send Data with No acknowledge (FDL Service)
SI	Serial Interface
SRD	Send and Request Data with reply (FDL Service)
SSAP	Source Service Access Point, an LSAP which identifies the local FDL User which initiates a transaction
Stn	Station, a device with an FDL Address on this PROFIBUS System
SYN	SYchroNizing bits of a frame (period of IDLE) it guarantees the specified frame integrity and allows for receiver synchronization
t_{BIT}	BIT Time, FDL symbol period the time to transmit one bit on this PROFIBUS System; 1/Data Signalling Rate in bit/s
T	Transmitter
$T_{A/R}$	Time to transmit an Acknowledgement/Response frame
TC	Token Cycle, cycle to transmit a token frame
T_{GUD}	GAP UpDate Time, the time this Master station waits between successive GAP maintenance cycles

T_{ID}	IDle Time, the time a Master station must wait between the last bit of a transmitted or received frame and the first bit of the next frame it transmits; this creates the interframe SYNchronizing period of IDLE
T_{MC}	Message Cycle Time, the time between transmission of the first bit of an action frame and receipt of the last bit of the corresponding reply frame
T_{QUI}	QUIet Time, Transmitter fall Time (Line State Uncertain Time) and/or Repeater switch Time; the time a transmitting station must wait after the end of a frame before enabling its receiver
T_{RDY}	ReaDY Time, the time after which the transmitting Master will reply frame
T_{RR}	Real Rotation Time, the time between the last successive receptions of the token by this Master station
TS	This Station
T_{SDI}	Station Delay of Initiator, the time this Master station will wait before sending successive frames
T_{SDR}	Station Delay of Responder, the actual time this responder waits before generating a reply frame
T_{SET}	SETup Time, the time between an event (e.g. interrupt SYN timer expired) and the necessary reaction (e.g. enabling receiver)
T_{SL}	SLot Time, the maximum time a Master station must wait for a transaction response
T_{SM}	Safety Margin Time
T_{SR}	System Reaction Time, the target maximum time to complete all transactions (including retries) in one poll cycle in single Master PROFIBUS Systems
$T_{S/R}$	Send/Request frame Time, the time to transmit a send/request frame
T_{SYN}	SYNchronization Time,the period of IDLE in front of frames after which this station enables its receiver; the required minimum inter-frame IDLE period to guarantee frame integrity and a valid frame
T_{SYNI}	SYNchronization Interval Time, the maximum time that a receiving station waits for the required inter-frame IDLE period, of duration T_{SYN}, to occur before it detects a bus fault
T_{TC}	Token Cycle Time, the time to pass the token to the next Master station in the absence of errors
T_{TD}	Transmission Delay Time, the maximum delay experienced between one station's transmitter and any other station's receiver
T_{TF}	Token Frame Time, the time to transmit the token frame
T_{TH}	Token Holding Time, the remaining period which this current Master station is allowed to initiate transactions during this token round
TTL	Transistor Transistor Logic

T_{TO}	Time-Out Time, the period of IDLE after which this Master station will claim the (lost) token, or any station will report a time-out fault to its FMA User, depends on this station's FDL Address
T_{TR}	Target Rotation Time, the anticipated time for one token round, including allowances for high and low priority transactions, errors and GAP maintenance
TTY	Tele-TYpe
TxD	Transmit Data signal (PHY, RS-485)
UART	Universal Asynchronous Receiver/Transmitter
UC	UART Character
UE	negative acknowledgement, remote User interface Error (L/M_status of the service primitive)
µP	Microprocessor
VDE	Verband Deutscher Elektrotechniker (Society of German Electronics Engineers)
VP	Voltage Plus
1 Chip MC	Single Chip MicroComputer

3.2 Basic Properties

The PROFIBUS Specification defines the technical and functional characteristics of a serial fieldbus which is aimed at the inter-connection of digital field devices or systems with low or medium performance, e.g. sensors, actuators, transmitters, programmable logical controllers (PLC), numerical controllers (NC), programming devices, local man machine interfaces etc.

Often a field control system is based on a central control and supervision unit, which is connected to a number of devices and small systems distributed in the field. In such cases the dominant data transfer is centrally oriented and cyclic from the field devices to the central data processing unit or to a superior control system.

The system contains master stations and slave stations. A master is able to control the bus, i.e. it may transfer messages without remote request when it has the right of access (called token). In contrast to this a slave is only able to acknowledge a received message or to transfer data after a remote request.

The token circulates in a logical ring formed by the masters. If the system contains only one master, e.g. a central control and supervision station, no token passing is necessary. This is a pure single-master/n-slave system. The minimum configuration comprises one master and one slave, or two masters.

The Physical Layer (this is the medium, including lengths and topology, the line interface, the number of stations and the transmission speed, variable in the range from 9,6 to 1500 kbit/s) can be adapted to different applications. However there is a common access method and transmission protocol and there are common services at the user interface.

3.3 Characteristic Features

The various application fields, e.g. process control, factory automation, power distribution, building automation, primary process industry etc., have the following characteristic bus requirements:

Network topology	Linear bus with or without terminator, including drop cables and branches (tree)
Medium, distances, number of stations	Depending on the signal characteristics, e.g. for shielded twisted pair, ≤ 1,2 km without repeaters, 32 stations
Transmission speed	Depending on network topology and line lengths, e.g. step-wise from 9,6 to 1500 kbit/s
Redundancy	Second medium is optional
Transmission characteristics	Halfduplex, asynchronous, slip protected synchronization (no bit stuffing)
Addressing	0 to 127 (127 = global addresses for broad-cast and multicast messages), address extension for regional address, segment address and Service Access address (Service Access Point, LSAP), 6 bit each
Station types	Masters (active stations, with bus access control); Slaves (passive stations, without bus access control); preferably at most 32 masters, optionally up to 127, if the applications are not time critical
Bus access	Hybrid, decentral/central; token passing between master stations and master-slave between master and slave stations
Data transfer services	Acyclic: Send Data with/without Acknowledge Send and Request Data with Reply Cyclic (Polling): Send and Request Data with Reply
Frame length	1 or 3 to 255 byte per frame, 0 to 246 layer 2 data octets for each Data Unit without address extension
Data integrity	Messages with Hamming distance (HD)=4, sync slip detection, special sequence to avoid loss and multiplication of data

3.4 Scope

This part of the specification deals with "Layer 0" and Layer 1 (Physical Media and Physical Layer PHY, according to the ISO-OSI Layer Model) which describe the medium and signal characteristics; furthermore Part 3 and 4 defines the Layer 2 (Fieldbus Data Link Layer, FDL), containing the transmission protocol and Medium Access Control, as well as the Interface Services (FDL/User Interface Services) used by the Layer 7 (Application Layer APP) immediately above. Finally the management of the Layers 1 and 2 (FMA1/2), including the user services, is covered (cf. Fig. 1).

In order to achieve high efficiency and throughput as well as low hardware and software costs the Layers 3 to 6 (Network, Transport, Session, Presentation) are left empty. A few relevant functionalities of these layers are realised in Layer 2 or in Layer 7.

The Application Layer functionalities (APP) and the related management (FMA7) are specified in Part 5, 6 and 7 of this specification.

It is intended to specify different Physical Layers to meet the requirements of various applications. All present and future variants use the same common Medium Access Control protocol and transmission protocol (formats, procedures), and they have a common interface to the Application Layer, including the related services.

The version 1 Physical Layer uses the US standard EIA RS-485. Further versions are to be specified in the future.

The octet (character) format is the UART format FT 1.2 (asynchronous transmission with start-stop synchronization) for Telecontrol Equipment and Systems, which is specified in IEC 870-5-1.

The transmission protocol definitions are close to the standard IEC 870-5-2, specified by IEC TC 57.

The Medium Access Control protocol, the data transfer services and the management services are guided by the standards DIN 19 241-2, IEC 955(PROWAY C), ISO 8802-2 and ISO/IEC JTC 1/SC 6N 4960 (LLC type 1 and LLC type 3). The complete protocol takes into consideration the high data integrity requirements of process data transmission.

Appendix A explains possible structures of repeaters and fieldbus devices as well as a topology with a single master station serving several fieldbus lines. Some guidelines on a redundant central control unit and a bus analyser/ diagnostic unit are added. Finally the data transfer rate and the system reaction time is calculated for an example.

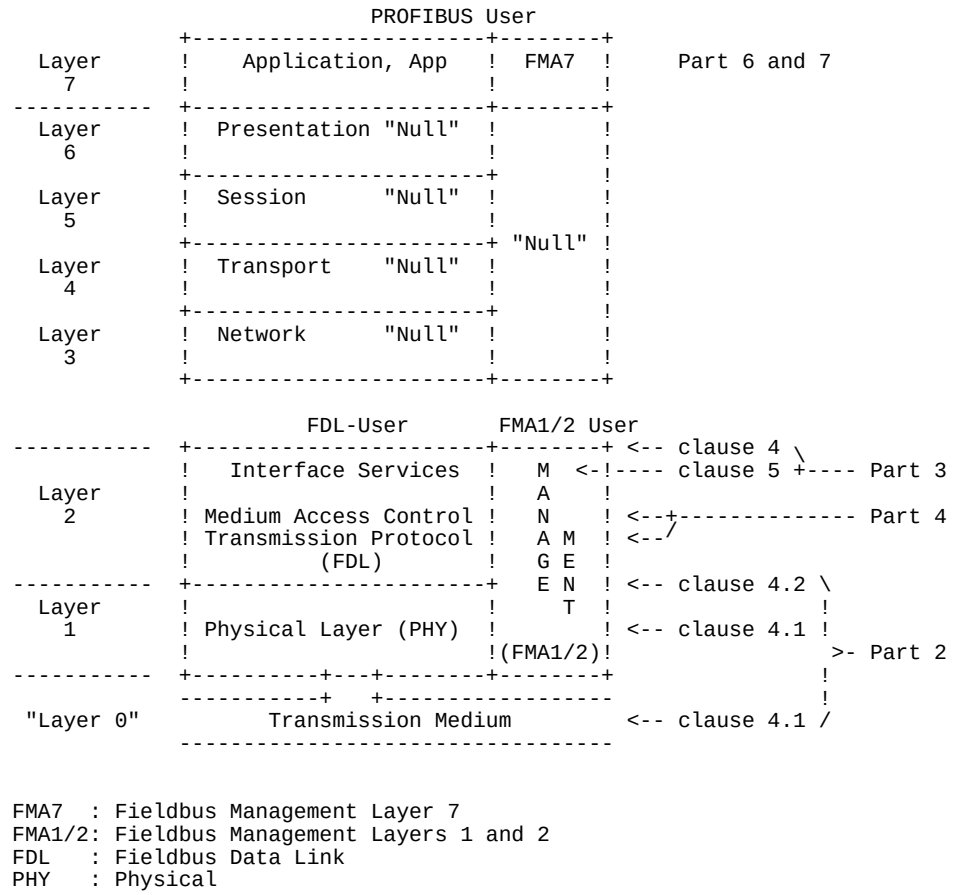


Figure 1. ISO Layer Model

4 Data Transmission (Physical Media, Physical Layer)

In order to cover a variety of requirements regarding topology, line length, number of stations, data transfer rate and protection against environmental influences, several Physical Layer versions are supported:

Version 1:

NRZ bit encoding is combined with EIA RS-485 signalling, targeted to low cost line couplers, which may or may not isolate the station from the line (galvanic isolation); line terminators are required, especially for higher date rates (up to 1500 kbit/s).

Version 2:

Flexible topology, covering a large area (tree topology), line couplers which consume less power and which reduce the influence of defective stations on bus operation, power transmission via the signal conductors, explosive atmosphere protection (Intrinsic Safety) and improved electromagnetic compatibility, a Physical Layer conform to IEC 1158-2.

The future versions shall be based on the Layer 1/Layer 2 interface described in clause 4.2, which includes definitions of the service primitives and the related parameters as well as the recommendations concerning encoding and signalling.

4.1 Version 1

The version 1 specifications describe a balanced line transmission corresponding to the US standard EIA RS-485 (EIA: Electronic Industries Association; RS-485: Standard for electrical characteristics of generators and receivers for use in balanced digital multipoint systems). Terminators, located at both ends of the twisted pair cable, enable the version 1 Physical Layer to support in particular higher speed transmission. The maximum cable length is 1,2 km for data rates ≤ 93,75 kbit/s. For 1500 kbit/s the maximum length is reduced to 70/200 m for Type B/A cable (see subclause "Bus Cable").

4.1.1 Electrical Characteristics

- Topology : Linear bus, terminated at both ends, stubs ≤ 0,3 m, no branches;
  In contrast to the EIA RS-485 recommendations it is good practice to allow longer stubs, if the total of the capacities of all stubs (Cstges) does not exceed the following values:
  Cstges ≤ 0,2 nF @ 1500 kbit/s
  Cstges ≤ 0,6 nF @ 500 kbit/s
  Cstges ≤ 1,5 nF @ 187,5 kbit/s
  Cstges ≤ 3,0 nF @ 93,75 kbit/s
  Cstges ≤ 15 nF @ 9,6 and 19,2 kbit/s
  It shall be taken into consideration that the total line length includes the sum of the stub lengths.

- Medium : Shielded Twisted Pair, see subclause "Bus Cable"

- Line Length : ≤ 1200 m, depending on the data rate and cable type

- Number of stations : 32 (master stations, slave stations or repeaters)

- Data rates : 9,6 / 19,2 / 93,75 / 187,5 / 500 / 1500 kbit/s, additionally higher data rates can be supported.

- Transceiver chip : e.g. SN 75176A, DS3695 or others

The line length and number of connected stations may be increased by using repeaters (bidirectional amplifiers, cf. annex 2-A.1). A maximum of 3 repeaters between two stations is permissible. If the data rate is ≤ 93,75 kbit/s and if the linked sections form a chain (linear bus topology, no active star) the maximum permissible topology assuming wire size 0,22 mm^2 (24 AWG, **A**merican **W**ire **G**auge) is as follows:

1 repeater : 2,4 km and 62 stations

2 repeaters: 3,6 km and 92 stations (cf. Fig. below)

3 repeaters: 4,8 km and 122 stations

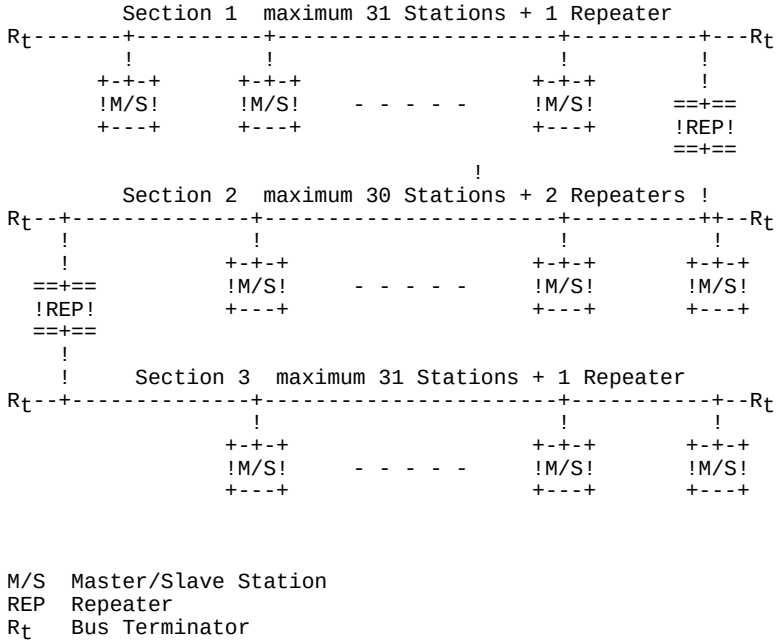


Figure 2. Repeater in Linear Bus Topology

In a tree topology more than 3 repeaters may be used and more than 122 stations may be connected, e.g. 5 repeaters and 127 stations. A large area may be covered by this topology, e.g. 4,8 km length and data rate ≤ 93,75 kbit/s with wire size 0,22 mm^2.

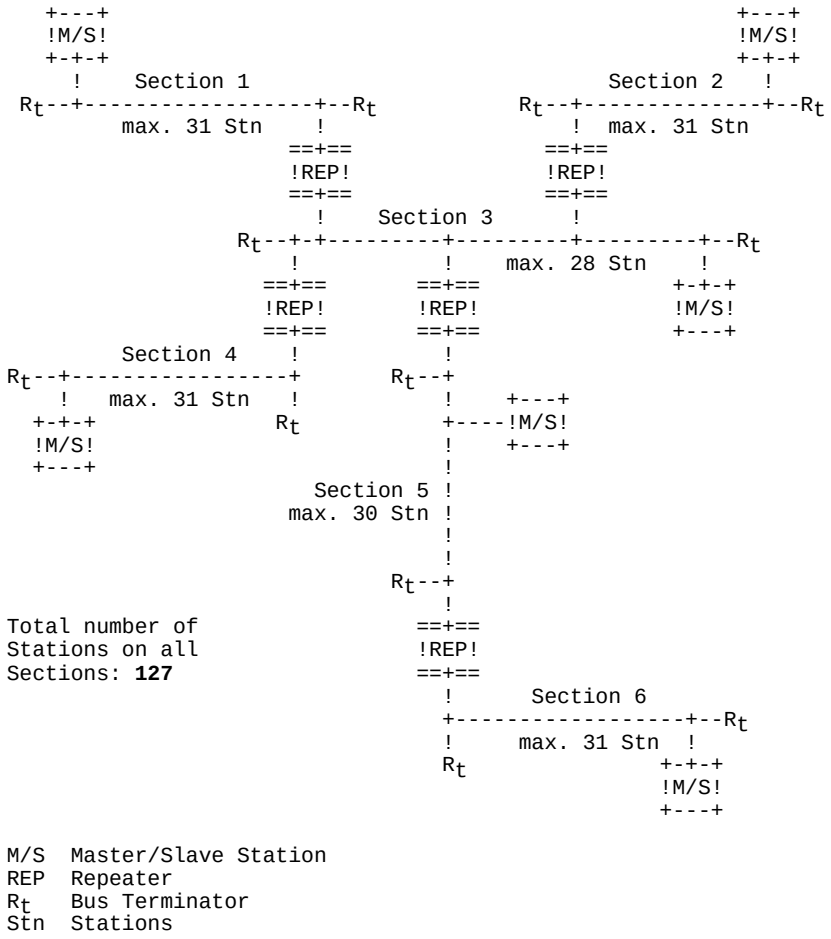


Figure 3. Repeater in Tree Topology

4.1.2 Connector Technique, Mechanical and Electrical Specifications

4.1.2.1 Bus Connector

Each station is connected to the medium via a 9-pin D-sub connector. The female side of the connector is located in the station, while the male side is mounted to the bus cable.

The mechanical and electrical characteristics are specified in IEC 807-3.

Preferably a metal connector housing should be used. When put together both parts of the connector should be fixed by conducting screws.

The connection between the cable sections and the stations should be realised as T-connectors, containing three 9-pin D-sub connectors (two male connectors and one female connector). Such T-connectors allow disconnection or replacement of stations without cutting the cable and without interrupting operation (on line disconnection).

4.1.2.2 Contact Designations

The pin assignments for the connectors are shown in Table 1.

Table 1. Contact designations

Pin No	RS-485 Ref.	Signal Name		Meaning
1		SHIELD	2)	Shield, Protective Ground
2		M24V	2)	Minus 24V Output Voltage
3	**B/B'**	**RxD/TxD-P**		**Receive/Transmit-Data-P**
4		CNTR-P	2)	Control-P
5	**C/C'**	**DGND**		**Data Ground**
6		VP	1)	Voltage-Plus
7		P24V	2)	Plus 24V Output Voltage
8	**A/A'**	**RxD/TxD-N**		**Receive/Transmit-Data-N**
9		CNTR-N	2)	Control-N

1) Signal is only necessary at station at end of the bus cable
2) Signals are optional

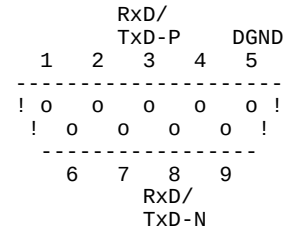


Figure 4. Connector pinout, front view of male and back view of female respetively.

The Data Ground, connected to pin 5, and the Voltage Plus, connected to pin 6, supply the Bus Terminator (cf. subclause 4.1.2.5).

The control signals, connected to pin 4 and pin 9, support direction control when repeaters without self control capability are used. RS-485 signalling is recommended. For simple devices the signal CNTR-P may be a TTL signal (1 TTL load) and the signal CNTR-N may be grounded (DGND). The definition of signalling is not subject of this specification.

The 24V output voltage, according to IEC 1131-2, allows the connection of operator panels or service devices without integrated power supply. If a device offers P24V this P24V shall allow a current load up to 100 mA.

4.1.2.3 Bus Cable

The PROFIBUS (version 1) medium is a shielded twisted pair cable. The shield helps to improve the electromagnetic compatibility (EMC). Unshielded twisted pair may be used, if there is no severe electromagnetic interference (EMI).

The characteristic impedance of the cable shall be in the range between 100 and 220 Ω, the cable capacity (conductor - conductor) should be < 60 pF/m and the conductor cross sectional area should be ≥ 0,22 mm^2 (24 AWG). Cable selection criteria are included in the appendix of the US standard EIA RS-485.

Two types of cables are defined:

Table 2. Cable specifications

Cable Parameter	Type A	Type B
Impedance	135 to 165 Ω (f = 3 to 20 MHz)	100 to 130 Ω (f > 100kHz)
Capacity	< 30 pF/m	< 60 pF/m
Resistance	< 110 Ω/km	-
Conductor area	≥ 0,34 mm^2 (22 AWG)	≥ 0,22 mm^2 (24 AWG)

The following table shows the maximum length of cable Type A and cable Type B for the different transmission speeds.

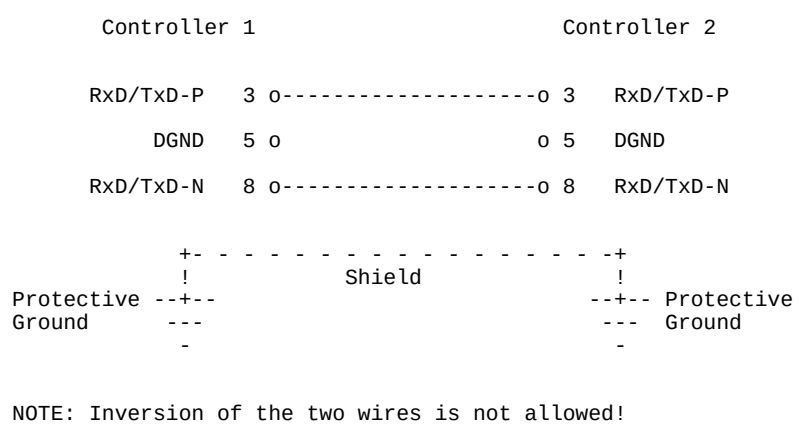
Table 1. Cable length for the different transmission speeds

Baud rate [kbit/s]	9.6	19.2	93.75	187.5	500	1500
Cable Type A Length in m	1200	1200	1200	1000	400	200
Cable Type B Length in m	1200	1200	1200	600	200	70

The dependency of the permissible data rate upon the network expanse (maximum distance between two stations) is shown in Fig. 2-A.1 of the US standard EIA RS-422-A (also included in DIN 66 259 and CCITT V.11).

Note: The recommendations concerning the line length presume a maximum signal attenuation of 6 dB. Experience shows that the distances may be doubled if conductors with an area ≥ 0,5 mm^2 (20 AWG) are used.

The minimum wiring between two stations is shown in Fig. 5.



NOTE: Inversion of the two wires is not allowed!

Figure 5. Interconnecting Wiring

The wiring shown in Fig. 5 allows a common mode voltage between both stations (i.e. the voltage difference between the Protective Grounds) of at most ± 7 V. If a higher common mode voltage is expected, a compensation conductor between the grounding points shall be installed.

4.1.2.4 Grounding, Shielding

If a shielded twisted pair cable is used it is recommended to connect the shield to the Protective Ground at both ends of the cable via low impedance (i.e. low inductance) connections. This is necessary to achieve a reasonable electromagnetic compatibility.

Preferably the connections between the cable shield and the Protective Ground (e.g. the metallic station housing) should be made via the metallic housings and the metallic fixing screws of the sub-D connectors. If this is not possible the pin 1 of the connectors may be used.

Grounding and shielding shall be according to DIN/VDE 0160 and DIN 57 899/ VDE 0800.

4.1.2.5 Bus Terminator

The bus cable Type "A" and "B" shall be terminated at both ends with R_{tA} respectively R_{tB}. The termination resistor R_t specified in EIA-RS-485 shall be complemented by a pulldown resistor R_d (connected to Data Ground DGND) and by a pullup resistor R_u (connected to Voltage-Plus VP). This supplement forces the differential mode voltage (i.e. the voltage between the conductors) to a well defined value when no station is transmitting (during the idle periods).

Each station which is destined to terminate the line (in common with a Bus Terminator) shall make Voltage-Plus (e.g. + 5 V ± 5%) available at pin 6 of the bus connector.

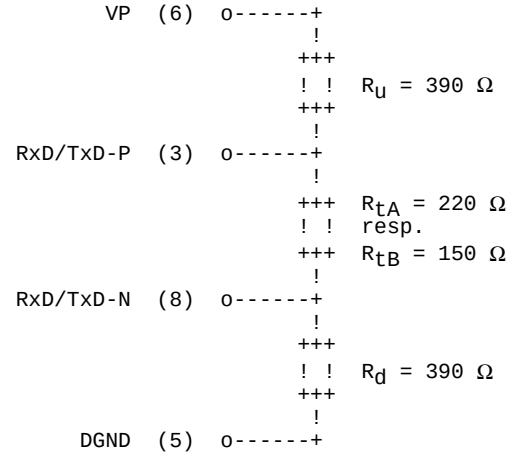


Figure 6. Bus Terminator

Assuming a power supply voltage of + 5 V ± 5% the following resistor values are recommended:

R_{tA} = 220 Ω ± 2%, min. 1/4 W;

R_{tB} = 150 Ω ± 2%, min. 1/4 W;

$R_u = R_d$ = 390 Ω ± 2%, min. 1/4 W

The power source supplying pin 6 (VP) shall be able to deliver a current of at least 10 mA within the specified voltage tolerances.

A mixture of both cable types and cable termination resistors as described above is allowed for a PROFIBUS System. However, the maximum line length has to be reduced up to the half of the above fixed values if line termination and line impedance do not match.

4.1.3 Transmission Method

4.1.3.1 Bit Encoding

In the version 1 of the Physical Layer of the PROFIBUS Specification NRZ (Non Return to Zero) coded data is transmitted via a twisted cable. A binary "1" is represented by a constant positive differential voltage between pin 3 (RxD/TxD-P) and pin 8 (RxD/TxD-N) of the bus connector, a binary "0" by a constant negative differential voltage.

4.1.3.2 Transceiver Control

When a station is not transmitting the transmitter output shall be disabled, it shall present a high impedance to the line. Preferably this should be controlled by the "Request to Send" signal (RTS, cf. clause 4.2). During the idle periods, i.e. when no data is transmitted by any station, the line signal shall represent a binary "1". Therefore the Bus Terminator shall force the differential voltage between the connector pins 3 and 8 to be positive when all transmitters are disabled. The line receivers shall always be enabled, therefore during idle the

binary signal "1" is received by every station (cf. Part 4, subclause 4.1.7, Syn time).

4.2 Interface between Physical Layer (PHY) and Medium Access and Transmission Protocol (FDL)

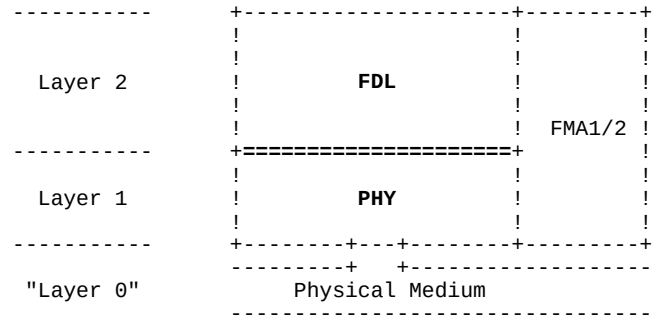


Figure 7. Interface between PHY and FDL in Relation to Layer Model

This clause includes an abstract description of the PHY data service. The service is provided by the PHY Layer to the FDL Layer. It supports reception and transmission of bits (FDL symbols) which are elements of an UART character (cf. Part 4, subclause 4.5.1). Each FDL symbol is of duration one bit time t_{BIT} (cf. Part 4, subclause 4.1.7).

The interface description does not specify or constrain the implementation within a fieldbus entity.

4.2.1 Overview of the Interaction

The PHY data service includes two service primitives. A request primitive is used to request a service by the FDL controller; an indication primitive is used to indicate a reception to the FDL controller. The names of the respective primitives are as follows:

PHY_DATA.request

PHY_DATA.indication

Temporal relationship of the primitives:

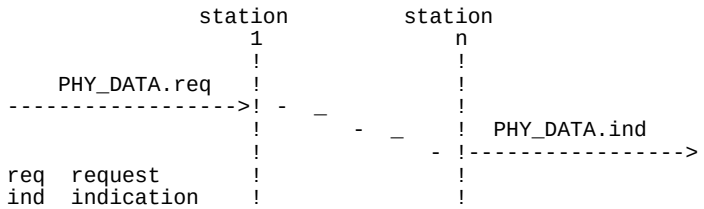


Figure 8. PHY Data Service

4.2.2 Detailed Specification of the Service and Interaction

This subclause describes in detail the service primitives and the related parameters in an abstract way. The parameters contain information needed by the Physical Layer entity.

Parameters of the primitives:

PHY_DATA.request
(FDL_symbol)

- The parameter FDL_symbol shall have one of the following values:

  a) ZERO corresponds to a binary "0"

  b) ONE corresponds to a binary "1"

  c) SILENCE disables the transmitter when no valid FDL symbol is to be transmitted

The PHY_DATA.request primitive is passed from the FDL Layer to the PHY Layer to request that the given symbol shall be sent to the fieldbus medium.

The reception of this primitive causes the PHY Layer to attempt encoding and transmission of the FDL symbol using signalling according to the related PHY Layer specifications (e.g. according to version 1, clause 4.1).

The PHY_DATA.request is a timed request, which may only be made once per FDL symbol period (t_{BIT}). The PHY Layer may confirm this primitive with a locally defined confirmation primitive.

PHY_DATA.indication
(FDL_symbol)

- The parameter FDL_symbol shall have one of the following values:

  a) ZERO corresponds to a binary "0"

  b) ONE corresponds to a binary "1"

The PHY_DATA.indication primitive is passed from the PHY Layer to the FDL Layer to indicate that a FDL symbol was received from the fieldbus medium. The effect of receipt of this primitive by the FDL Layer is not specified.

The PHY_DATA.indication is a timed indication, which may be made only once per received FDL symbol period (t_{BIT}).

4.2.3 Electrical Requirements and Encoding

The physical characteristics of the interface between PHY and FDL Layer (connector type, pin assignment, driver and receiver characteristics etc.) are not specified. Experience shows that at least the following three signals are required: Transmitted Data (TxD), Received Data (RxD) and Transmitter Enable (Request to Send, RTS). An example of signal encoding is as follows:

PHY_DATA.request parameter values	PHY transmit encoding TxD	RTS
ZERO	0	1
ONE	1	1
SILENCE	x	0

x: irrelevant

PHY_DATA.indication parameter values	PHY receive encoding RxD
ZERO	0
ONE	1

The FDL Layer generates its own time frame (FDL symbol period, UART clock). Consequently no time information (transmit/receive clock) is needed from the PHY Layer.

For certain implementations it may be useful to pass time information from the PHY Layer to the FDL Layer or vice versa. Furthermore an additional control signal passed from the FDL Layer to the PHY Layer may be advisable: the Clear to Send (CTS).

Both functions are not included in the interface specifications. If required they shall be locally defined.

4.3 Redundancy of Physical Layer and Media (optional)

The use of redundant PHY Layers is supported to improve the reliability of the fieldbus. When implemented, the redundant PHY Layer contains two separately installed media (bus a and bus b) and two complete transceivers (transmitter & receiver) per station. The redundancy architecture is shown in Fig. 9.

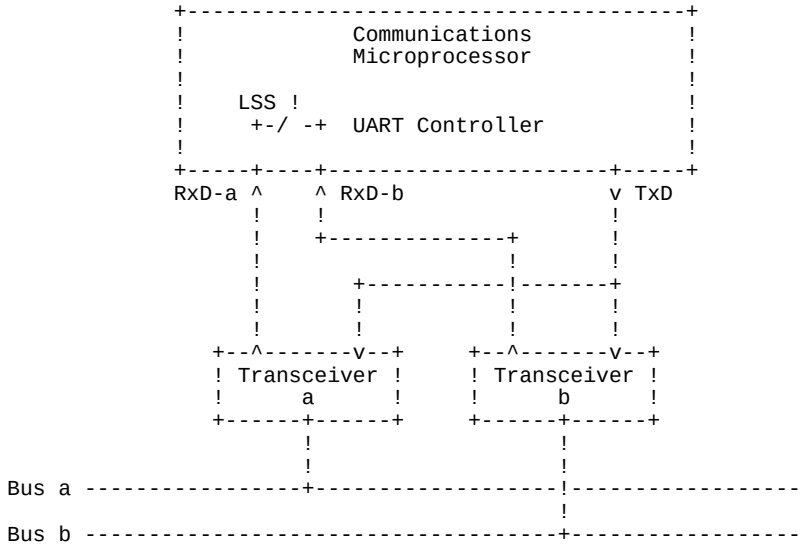


Figure 9. Redundancy of Physical Layer and Media

The basic principle, shown in Fig. 9, assumes that data is sent out simultaneously by both transceivers onto both media (Bus a and Bus b). In contrast to this each station receives from only one medium (either Bus a or Bus b). The receive channel is selected by a Line Selector Switch (LSS), which is located between both transceivers and the UART controller, or - if two UART controllers are used - between the UART controllers and the Communications Microprocessor. The Line Selector Switch is completely controlled by the FDL Layer. For this the Communications Microprocessor of each station monitors the medium activity, independently of any other station. The main switching conditions for masters and slaves are as follows:

- Two or more successive invalid frames are received, i.e. UART characters with invalid format, invalid parity bit or invalid frame check sequence are detected.

- Time out timer T_{TO} expires, cf. Part 4, subclause 4.1.7.

- No minimum idle period (i.e. line idle for at least T_{SYN}) was detected during one Synchronization Interval Time T_{SYNI}, cf. Part 4, subclause 4.1.7.

The selected receive channel a (primary) or b (alternate) is notified to the Fieldbus Management (FMA1/2, cf. Part 3, clause 4.2). The Fieldbus Management then provides this information to the local user via the FMA1/2 interface. There is no preferred receive channel after the system initialization is finished.

Annex 2-A (informative)

Examples of Realizations

2-A.1 Repeater

Two fundamental repeater structures are realizable:

The remotely controlled and the self-controlled repeater. For the remotely controlled repeater in the ground state there is a preferred direction of the signal transfer: it shall always point from the upmost line to the lines beneath. If a station or a line beneath sends a message then it shall reverse the direction of the signal transfer of the repeater before the beginning of the transmission and thereafter restore the original condition. In general this means that each line beneath needs in addition a control bus line (CNTR: control; wire pair). The circuit expense of the repeater totals three receiver (R) and three transmitter (T) devices (see Fig. below).

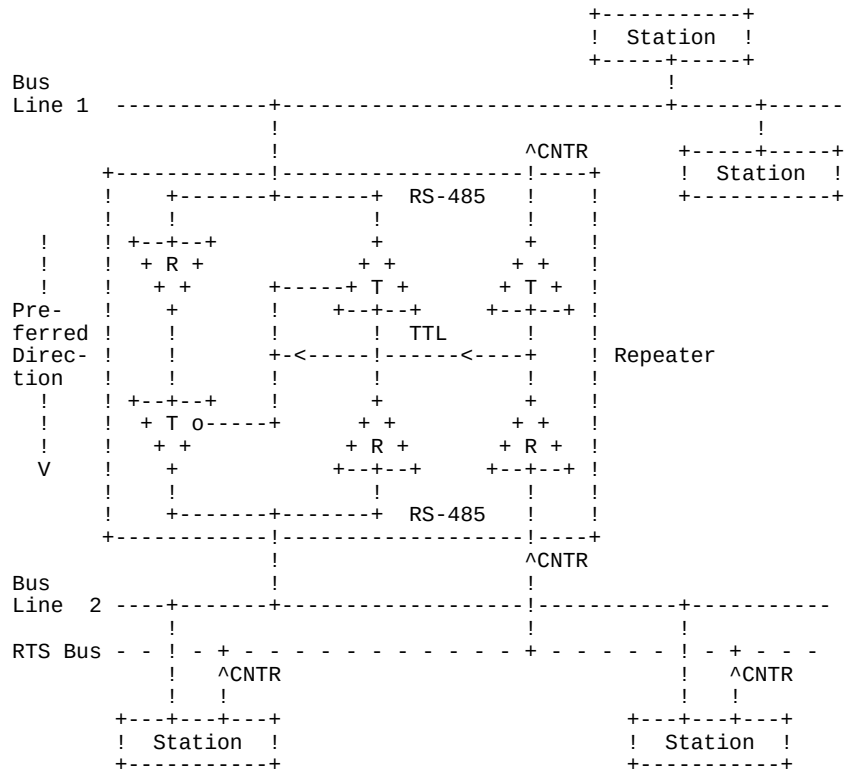


Figure 2-A.1 Remotely Controlled Repeater

For the self-controlled repeater in the ground state both bus lines are receivers. A signal transfer happens only upon arrival of a start bit edge in the corresponding direction. A priority circuit resolves conflicts if a start bit edge is received simultaneously from both sides. The detection of the end of a frame and then the reset into the ground state is carried out by a signal idle time watchdog (idle binary "1") of ≥ 11 bits. The circuit expense of the repeater comprises two receiver and transmitter devices each and one direction control with priority and idle time control (see Fig. below). The idle time control depends on the data signalling rate, and is set to be $\geq 11 \cdot t_{Bit}$.

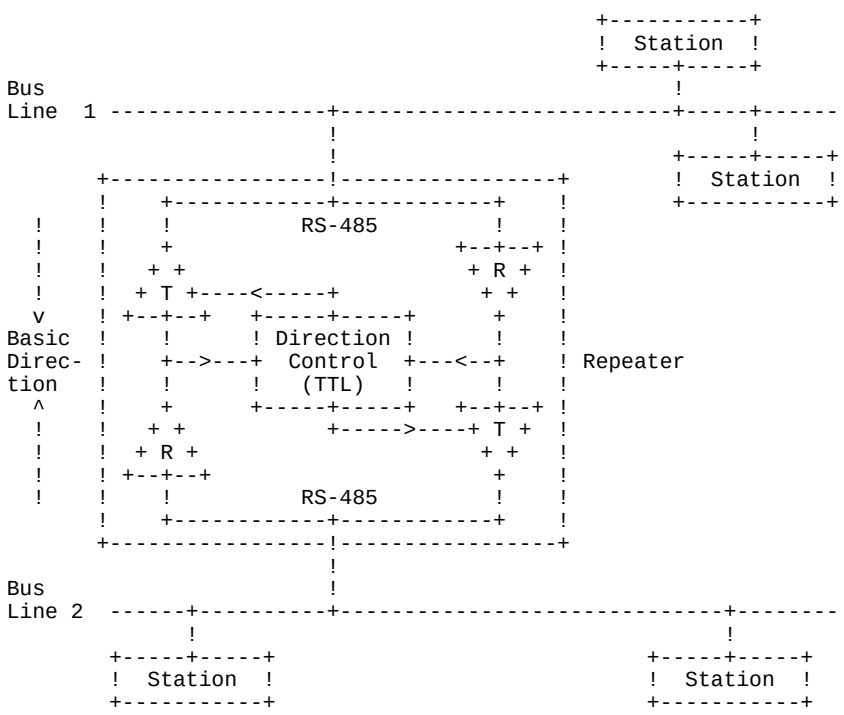


Figure 2-A.2 Self-Controlled Repeater

A temporal regeneration of the bit phases is not provided for either repeater version. Therefore the maximum number of successive repeaters is limited to three.

2-A.2 Structures of PROFIBUS Controllers

The PROFIBUS Controller (PBC) establishes the connection of a field automation unit (control or central processing station) or a field device to the transmission medium. The PBC consists of the line transmitter/receiver (Transceiver), the frame character transmitter/receiver (UART) and the FDL/APP processor with the interface to the PROFIBUS user. An electrical isolation may be inserted between the RS-485 transceiver and the UART with optical couplers as well as with other transmission techniques between the transmission medium and transceiver, e.g. with transmitters (see Fig. below).

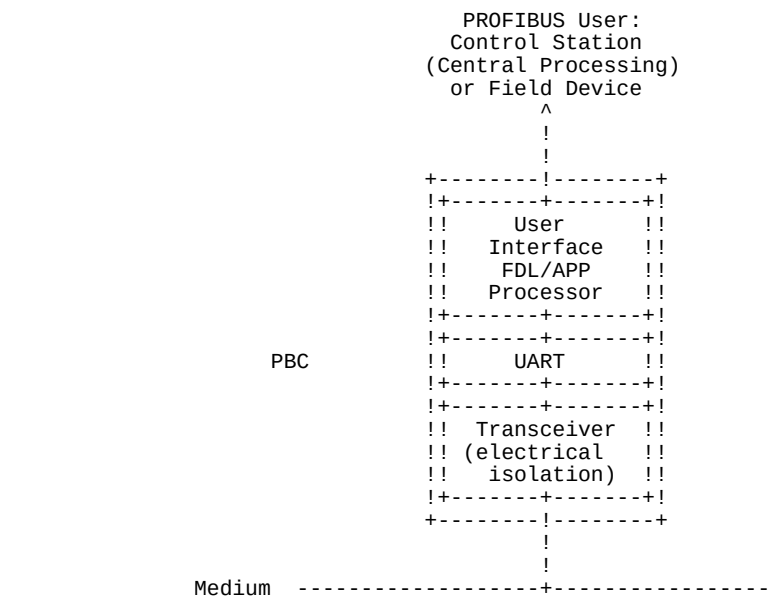


Figure 2-A.3 PROFIBUS Controller (PBC)

For the adaption to the different field automation units and devices mostly three variants of the PROFIBUS Controller (Controller_type) are used that differ distinctly in expense (see Fig. below):

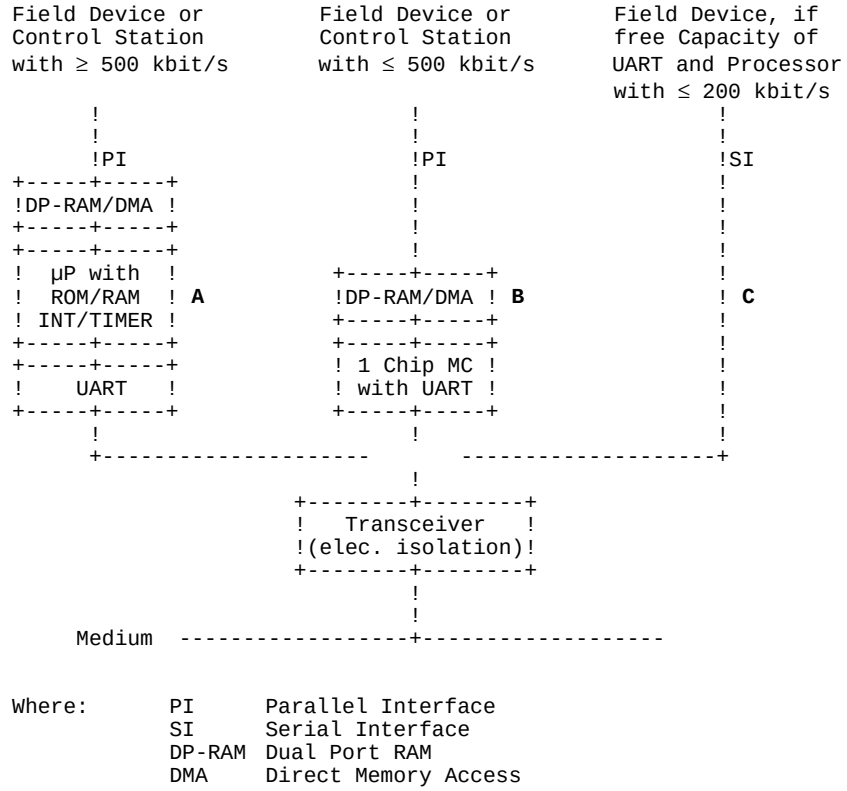


Figure 2-A.4 Variants of PBC (Controller_type)

A: For data signalling rates at, or in future also above, 500 kbit/s a suitable microprocessor (e.g. 80186) with adequate memory and peripheral devices becomes necessary for the FDL/APP processor. The interface to the user is realized preferably by a Dual Port RAM or a DMA.

B: For data signalling rates ≤ 500 kbit/s single chip microprocessors with UART device (e.g. 80C51FA, V25 [µPD 70322]) are now suitable. As above a Dual Port RAM or DMA is preferred for the user interface.

C The minimum expense results from shifting the FDL/APP protocol into the device CPU. For this a UART interface shall be free and free capacity in the device processor shall be available for the FDL/APP protocol. Often the obtainable data signalling rate is not above 200 kbit/s.

2-A.3 System with several Bus Lines to one Control Station

The division in several self-sufficient bus lines offers an alternative for pure master - slave systems, which need more than 32 stations in each system, and which cannot employ repeaters. These lines are operated in time multiplexing by one master station or in parallel with one master station each. For a layout of e.g. four lines according to version 1, 125 stations are permissible altogether. With regard to simplicity, availability and reaction time such a layout is especially favourable. In case of breakdown of one line the remaining stations are able to communicate with the control station, whereas for a single line all stations would be inaccessible.

Depending on the data signalling rate of the individual lines two structures exist that differ in expense. For up to about 200 kbit/s, a bus controller of PBC Type **A** as master station is able today to support 4 UART devices and therefore manage four bus lines, see following Figure:

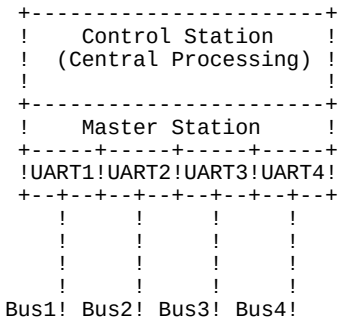


Figure 2-A.5 Several Buses controlled by one Master (PBC)

For data signalling rates above 200 kbit/s a separate bus controller as master station for each line becomes necessary. The number of buses possible depends on the capability of the control station, because it shall coordinate the masters (see Fig. below).

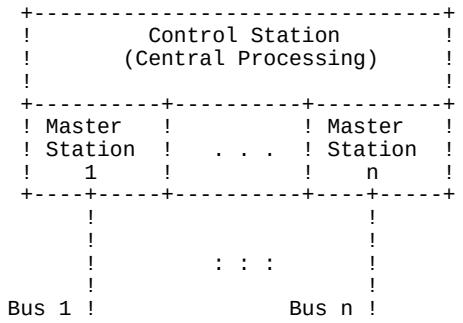


Figure 2-A.6 Several PBCs in one Control Station

2-A.4 Redundant Control Station

Redundancy of the complete control station is required to increase the availability of systems with only one master station (master - slave system). Merely a doubled bus controller would not provide the necessary availability to the control station in case of power failure. The basic layout may be with or without redundant transmission techniques (see Fig. below):

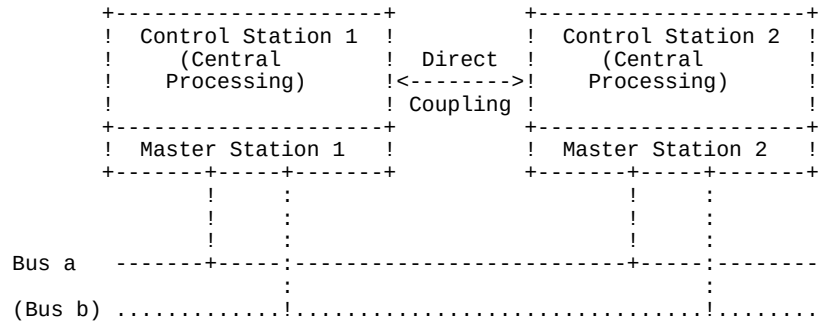


Figure 2-A.7 Redundant Control Station

In the above concept always only one control station is active, while the other is waiting in updated condition (hot standby). By means of activity control (time-out) of the bus lines the standby station is able to independently take over the control function upon failure of the control station. For this purpose it shall hold a current data record, which is exchanged between the units preferably by direct coupling. At system start-up the active control station is determined by prioritization.

2-A.5 Bus Analysis/Diagnostic Unit (Bus Monitor)

During start-up, maintenance and in case of failure a control station is able to perform operational tests only to a limited extent. Extensive statements about the state of the PROFIBUS System are possible only with an additional analyse/diagnostic device (bus monitor) which may be connected to the bus at any place in the field. This unit has three operating modes:

- analysis mode
- diagnostic mode
- control station mode (standard mode PHY/FDL/APP)

In analysis mode events are acquired and analyzed on the PROFIBUS in certain time intervals and according to different criteria. The device operates purely passively as a monitor of all messages and does not influence the bus operation.

The diagnostic mode serves to detect and localize errors, to determine the system configuration and system state and to provide a lucid display of the acquired and analyzed station and error conditions. The diagnostic mode requires the transmission of dedicated messages and the triggering of certain actions such as special error states, bus load or the self test in the other stations. This mode is only permissible for the master station.

In control station mode the central processing function is performed. It is essential for the retention of bus operation after shut-down or failure of the active control station. The central control function may be taken over thereby to a limited extent only (see conditions for redundant control stations). The control station mode may also be used for the stepwise start-up of new stations in the PROFIBUS System.

The Figure below shows roughly the layout of such an analysis/diagnostic device. A standard PROFIBUS coupling for master stations is used (Type **A**), with an analysis/diagnostic computer (e.g. PC) coupled to the user interface. A standard keyboard with screen serves to operate and display PROFIBUS functions and states. This computer has a standard serial interface (CCITT V.24, TTY) for logging system conditions.

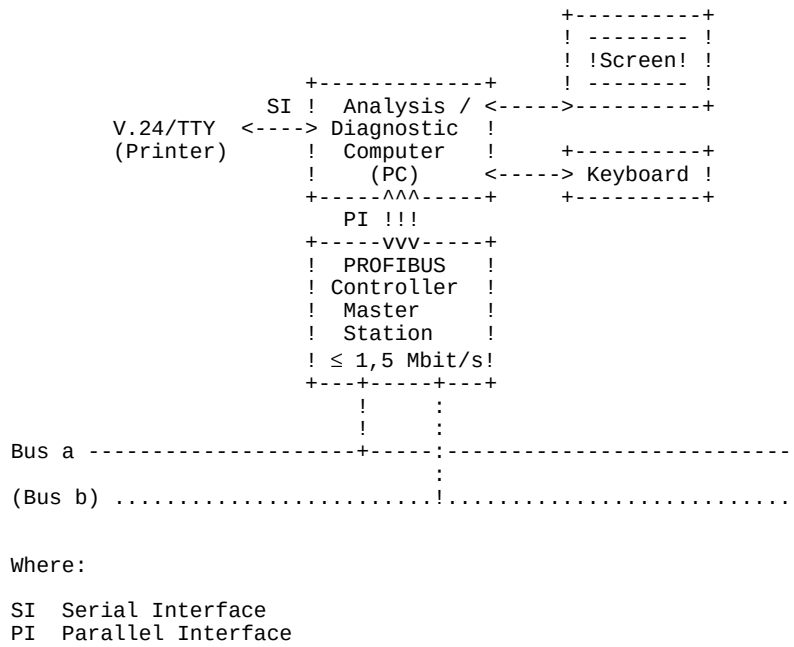


Figure 2-A.8 Bus Analysis/Diagnostic Unit

2-A.6 Performance of Message Rate, System Reaction Time and Token Rotation Time

The message rate R_{SYS} in the system corresponds to the possible number of message cycles per second (see Part 4, clause 4.2, formula (23)). The maximum system reaction time (also called station or bus request time) for cyclic send/request (polling) from one master station to n slave stations is computed from the message cycle time and the number of slave stations (see Part 4, clause 4.2, formula (24)).

For the following example calculation the times T are converted into t (seconds) and set as follows:

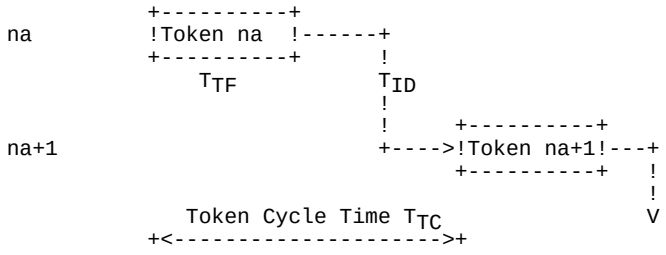
t_{SDR} = 0,5 ms, t_{ID} = 1 ms (or t_{SYN} + t_{SM}, if > 1 ms) and DATA_UNIT = 2, 10, 50 octet, assuming np = 30 slave stations. The request frame is in each case without DATA_UNIT (see Part 4, subclause 4.6.1A), while the response frame has a variable DATA_UNIT (see Part 4, subclause 4.6.3B).

The formulas (22), (23) and (24) described in Part 4, clause 4.2 yield the values listed in the table below for message cycle time, message rate and system reaction time versus data signalling rate, DATA_UNIT and number of slave stations. Message repetition and the time T_{TD} are not considered. Token transfer times are irrelevant, because there is only one master station in the system. The values are rounded. See also Fig. 2-A.11 and Fig. 2-A.12.

Table 2-A.1

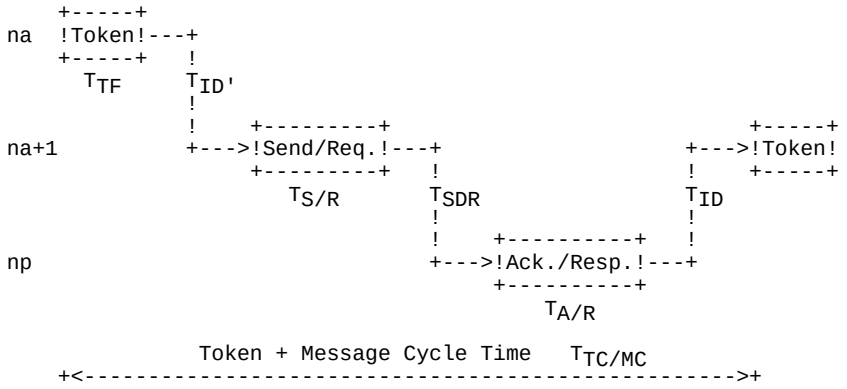
DATA_UNIT	Data Signalling Rate [kbit/s]					
	9,6	19,2	93,75	187,5	500	1500
	t_{MC} [ms] Time for One Message Cycle					
2 octets	24	12	3,5	2,5	1,9	1,6
10 octets	33	17	4,4	3	2,1	1,7
50 octets	79	40	9,2	5,3	2,93	2
	R_{SYS} [N/s] System Total Message Cycles/second					
2 octets	42	82	287	400	535	615
10 octets	30	60	226	337	488	594
50 octets	12	25	108	188	341	506
	t_{SR} [ms] System Reaction (Latency) Time with 30 Slave Stations					
2 octets	711	364	105	75	56	49
10 octets	986	502	133	89	62	51
50 octets	2,4s	1,2s	276	159	88	60

In a system with several master stations the system reaction time t_{SR} depends additionally on the real token rotation time t_{RR}. Fig. 2-A.13 shows the bounds of t_{RR} versus the number of master stations and high priority messages for 93,75, 500 and 1500 kbit/s. Message repetition and the time T_{TD} are not considered. The computation is based upon the token cycle time T_{TC} according to Fig. 2-A.9 and the combined token and message cycle time $T_{TC/MC}$ according to Fig. 2-A.10.



$$T_{TC} = T_{TF} + T_{ID} \; ; \; T_{TF} = 33 \text{ bit} \qquad (2\text{-A.1})$$

Figure 2-A.9 Cycle Time T_{TC}



$$T_{TC/MC} = T_{TF} + T_{ID'} + T_{S/R} + T_{SDR} + T_{A/R} + T_{ID} \qquad (2\text{-A.2})$$

Figure 2-A.10 Cycle Time $T_{TC/MC}$

Message Cycle:

Send/Request Data with Reply: $T_{S/R}$ = 66 bit

Response with 10 octet DATA_UNIT: $T_{A/R}$ = 209 bit

The following implementation dependent times were chosen:

Token Cycle: t_{ID} = 0,5 ms

Token + Message Cycle: t_{ID} = 0,5 ms ; $t_{ID'}$ = 1 ms ;
t_{SDR} = 0,5 ms

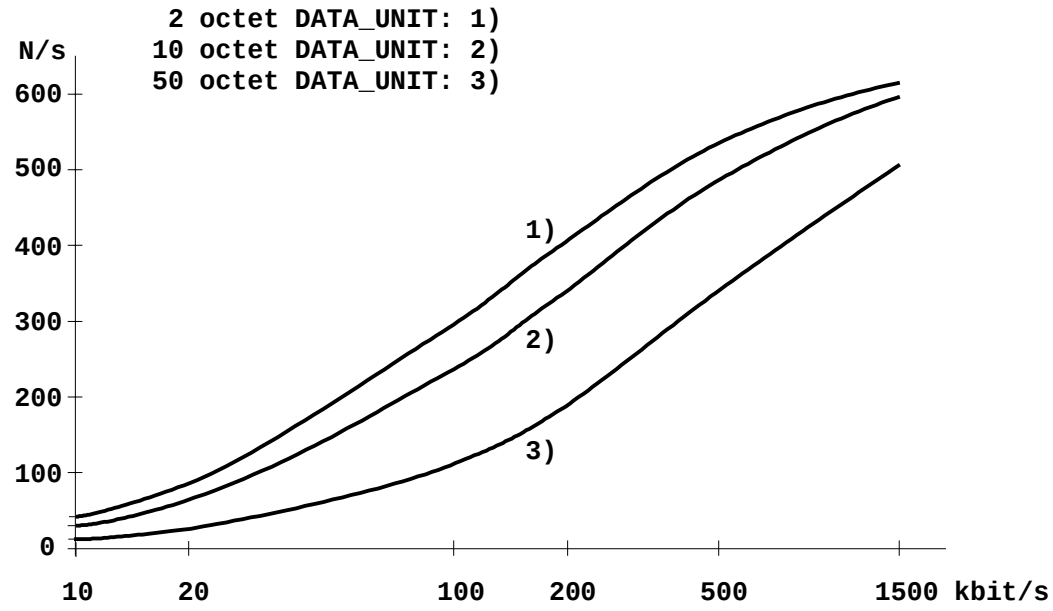


Figure 2-A.11 System Total Message Cycles R_{SYS}

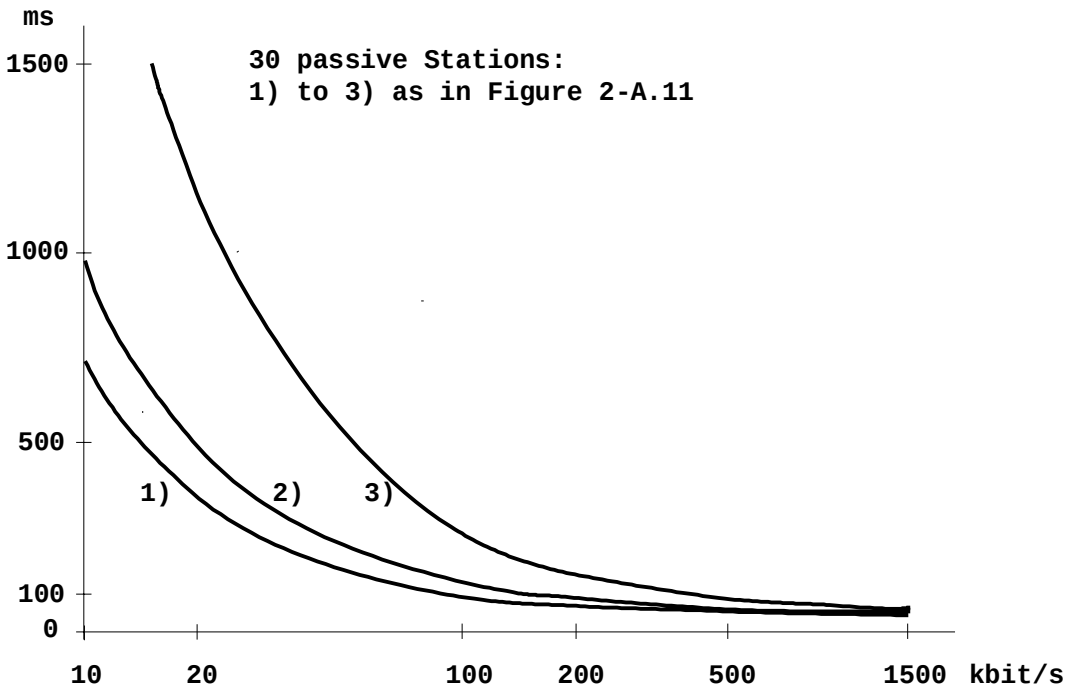


Figure 2-A.12 System Reaction Time t_{SR}

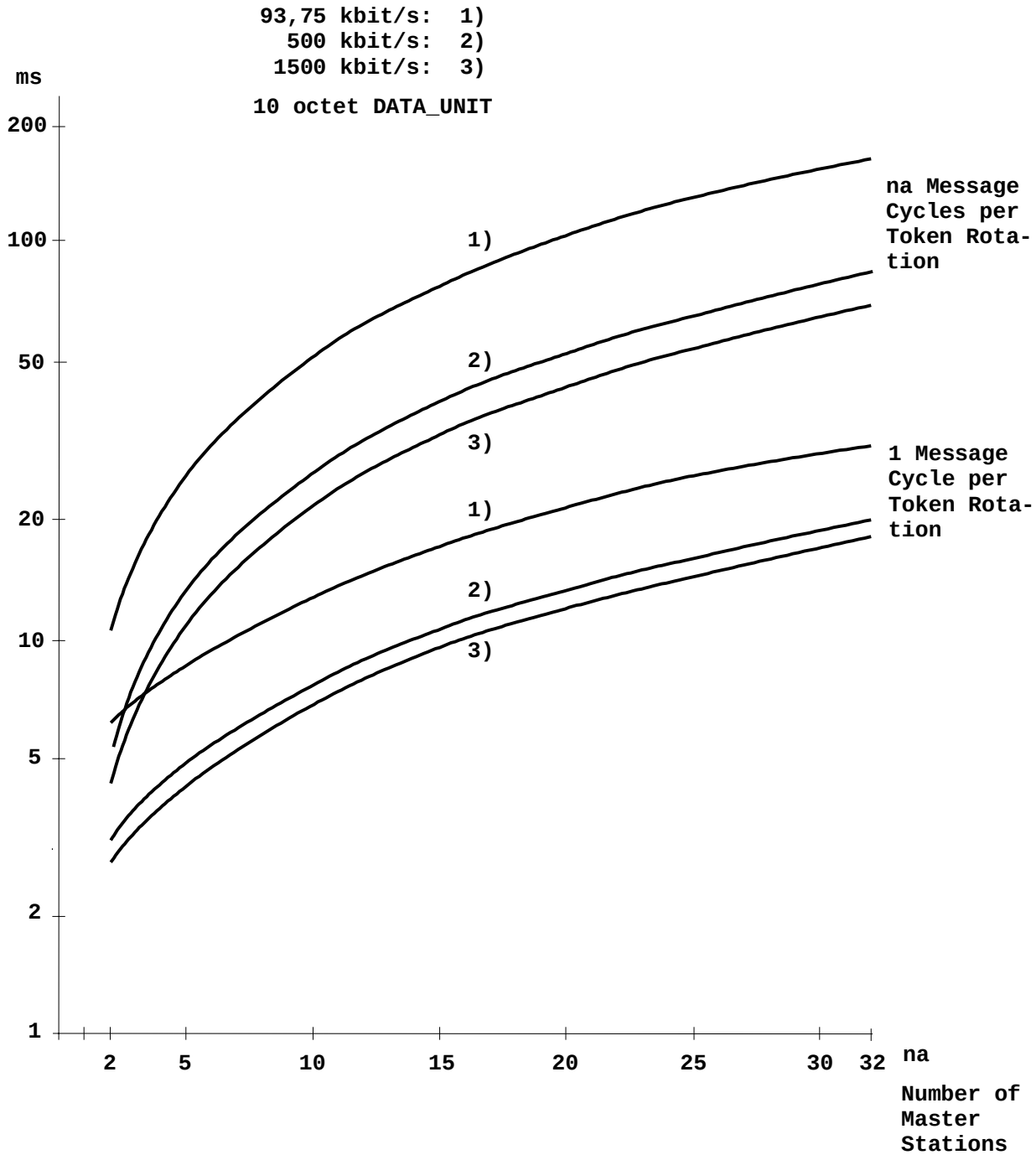


Figure 2-A.13 Real Token Rotation Time t_{RR}

PROFIBUS Specification - Normative Parts

Part 3

Data Link Layer Service Definition

Contents

1 Scope

(see Part 2)

2 Normative References

(see Part 2)

3 General

(see Part 2)

4 PROFIBUS Layer 2 Interface

The following subsections abstractly describe the data transfer (FDL = Fieldbus Data Link) and the management (FMA = Fieldbus Management) services of PROFIBUS. The FDL services are made available to the user via Layer 2. The FMA1/2 services are made available through the management (FMA1/2) associated with Layers 1 and 2. Neither the implementation of the controllers nor the hard-, firm- or software interfaces are specified or prescribed.

4.1 FDL User - FDL Interface

This clause describes the data transfer services available to the FDL user with their service primitives and their associated parameters. These services of the FDL are optional.

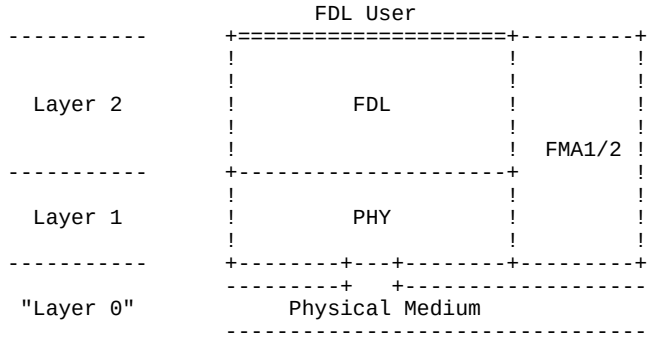


Figure 1. Interface between FDL User and FDL in Relation to Layer Model

4.1.1 Overview of Services

The user of Layer 2 is provided with the following data transfer services:

- Send Data with Acknowledge (SDA)
- Send Data with No Acknowledge (SDN)
- Send and Request Data with Reply (SRD)
- Cyclic Send and Request Data with Reply (CSRD)

Send Data with Acknowledge (SDA)

This service allows a user of the FDL (Layer 2) in a master station (called Local User in the following), to send user data (Link_service_data_unit, L_sdu) to a single remote station. At the remote station the L_sdu, if received error-free, is delivered by the FDL to the user (called Remote User in the following). The Local User receives a confirmation concerning the receipt or non-receipt of the user data. If an error occurred during the transfer, the FDL of the Local User shall repeat the data transfer.

Send Data with No Acknowledge (SDN)

This service allows a Local User to transfer data (L_sdu) to a single remote station, to many remote stations (Multicast), or to all remote stations (Broadcast) at the same time. The Local User receives a confirmation acknowledging the end of the transfer, but not whether the data was duly received. At the remote stations this L_sdu, if received error-free, is passed to the Remote User. There is no confirmation, however, that such a transfer has taken place.

Send and Request Data with Reply (SRD)

This service allows a Local User to transfer data (L_sdu) to a single remote station and at the same time to request data (L_sdu) that was made available by the Remote User at an earlier time. At the remote station the received L_sdu, if error-free, is passed to the Remote User. The service also allows a Local User to request data from the Remote User without sending data (L_sdu = Null) to the Remote User.

The Local User receives either the requested data or an indication that the data was not available or a confirmation of the non-receipt of the transmitted data. The first two reactions also confirm the receipt of the transferred data.

If an error occurs during the transfer, the FDL of the Local User repeats the data transfer with the data request.

Cyclic Send and Request Data with Reply (CSRD)

This service allows a Local User to cyclically transfer data (L_sdu) to a remote station and at the same time to request data from the remote station. At the remote station the data received error-free is passed cyclically to the Remote User. The service also allows a Local User to cyclically request data from the Remote User without sending data to the Remote User.

The Local User cyclically receives either the requested data or an indication that the data was not available or a confirmation of the non-receipt of the transmitted data. The first two reactions also confirm the receipt of the transferred data.

If an error occurs during the transfer, the FDL of the Local User repeats the data transfer with the data request.

The selected remote stations and the number and sequence of the data transfers with data requests for the cyclic mode shall be defined by the Local User in the Poll List.

4.1.2 Overview of Interactions

The services are realized by using a number of service primitives (denoted by FDL_...). To request a service the user employs a Request primitive. A Confirmation primitive is returned to the user upon completion of a service, or in the case of services with cyclic repetition, after every send/request cycle. If an unexpected event occurs at the remote station, the Remote User is informed by an Indication primitive. For the services mentioned above the following primitives apply:

	Possible for the following stations
Send Data with Acknowledge (SDA)	
FDL_DATA_ACK.request	Master
FDL_DATA_ACK.indication	Master and Slave
FDL_DATA_ACK.confirm	Master
Send Data with No Acknowledge (SDN)	
FDL_DATA.request	Master
.indication	Master and Slave
.confirm	Master
Send and Request Data with Reply (SRD)	
FDL_DATA_REPLY.request	Master
.indication	Master and Slave
.confirm	Master
FDL_REPLY_UPDATE.request	Master and Slave
.confirm	- " -
Cyclic Send and Request Data with Reply (CSRD)	
FDL_SEND_UPDATE.request	Master
.confirm	- " -
FDL_CYC_DATA_REPLY.request	Master
.confirm	- " -
FDL_CYC_ENTRY.request	Master
.confirm	- " -
FDL_CYC_DEACT.request	Master
.confirm	- " -
FDL_DATA_REPLY.indication	Master and Slave
FDL_REPLY_UPDATE.request	Master and Slave
.confirm	- " -

confirm: confirmation

Temporal Relationships of Service Primitives:

Notation in the following figures:

```
 .req   .request
 .ind   .indication
 .con   .confirmation

L_sdu   Link_service_data_unit
L_pci   Link_protocol_control_information
L_pdu   Link_protocol_data_unit
L_pdu   L_pci + L_sdu
```

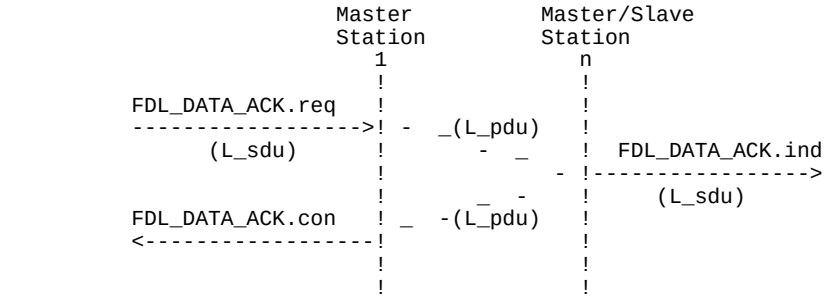


Figure 2. SDA Service

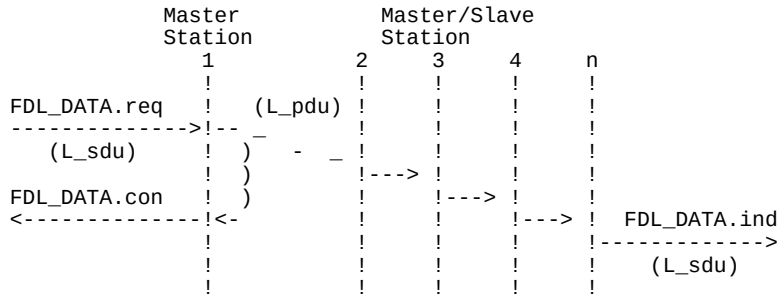


Figure 3. SDN Service

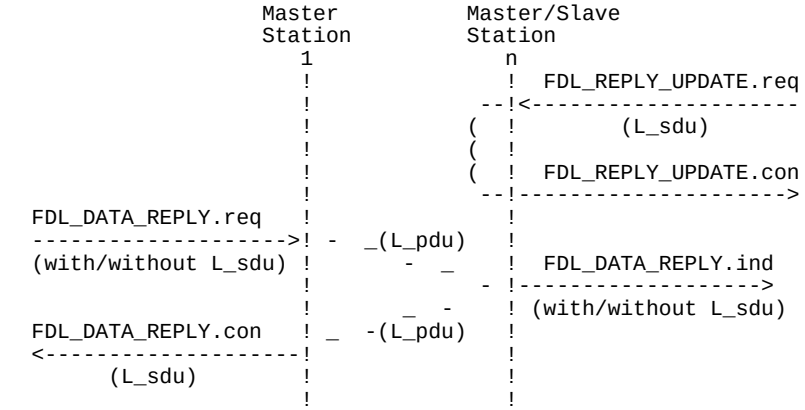


Figure 4. SRD Service

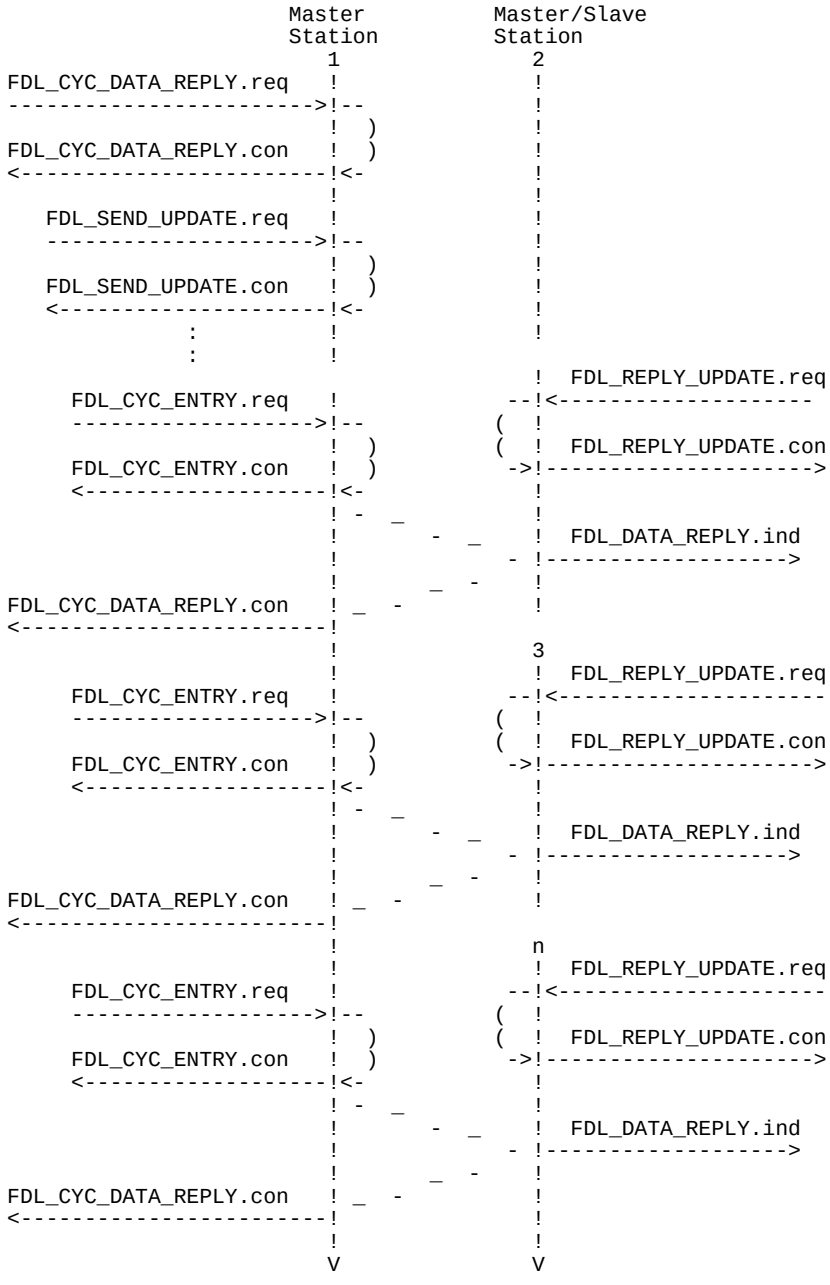


Figure 5a. Start of CSRD Service

Figure 5b. End of CSRD Service

4.1.3 Detailed Specification of Services and Interactions

4.1.3.1 Send Data with Acknowledge (SDA)

The Local User prepares a Link_service_data_unit (L_sdu) for the Remote User that contains transparent process information (e. g. data or commands) from the PROFIBUS for the FDL and PHY. This data is passed to the local FDL controller via the FDL interface by means of a FDL_DATA_ACK.request primitive.

The FDL controller accepts the service request and tries to send the L_sdu to the remote FDL controller (for the message format see Part 4, Table 3a, b7=1, Code No 3/5). The local FDL controller passes a confirmation to the Local User by means of the FDL_DATA_ACK.confirm primitive that signals either correct or erroneous data transfer.

Prior to passing a confirmation to the Local User, the local FDL controller needs an acknowledgement from the remote FDL controller (see Part 4, Table 3a, b7=0, Code No 0/1/2/3 or SC = E5H). If this acknowledgement is not received within the Slot Time T_{SL} (see Part 4, subclause 4.1.7), the local FDL controller shall try again (Retry) to send the L_sdu to the remote FDL. If no acknowledgement is received after n retries (max_retry_limit), the local FDL controller shall signal a negative acknowledgement to the Local User. During the transfer of the data and the receipt of the associated acknowledgement, no other traffic takes place on the PROFIBUS.

If the data frame was received error-free, the remote FDL controller passes the L_sdu to the Remote User via the FDL user interface by means of a FDL_DATA_ACK. indication primitive.

Parameters of the primitives:

FDL_DATA_ACK.request
(SSAP, DSAP, Rem_add, L_sdu, Serv_class)

- The parameter SSAP (Source Service Access Point) defines the service access point of the Local User. The Local User Access Address (see Part 4, subclause 4.8.2.2) identifies the SSAP. The SSAP value 63 (Global Access Address) is not permissible.

- The parameter DSAP (Destination Service Access Point) defines the service access point of the Remote User. The Remote User Access Address (see Part 4, subclause 4.8.2.2) identifies the DSAP.

PROFIBUS stations that do not carry LSAPs in the frames for efficiency reasons, set both the parameters SSAP and DSAP to NIL; this means that the Default LSAP is addressed (see Part 4, subclause 4.8.2.2).

- The parameter Rem_add (Remote_address) defines the FDL address of the remote station (Destination Address DA, DAE[b7=1], as defined in Part 4, subclause 4.8.2). Rem_add needs to be an individual address. The global address is not permissible.

- The parameter L_sdu (Link_service_data_unit) contains the user data that is to be transferred by the FDL controller. The L_sdu contains 1 octet as the minimum data unit, the maximum length is 246 octets. When using SSAP, DSAP and Region/Segment Addresses, a maximum of 242 octets is possible.

- The parameter Serv_class defines the FDL priority for the data transfer. Two priorities are possible (see Part 4, subclause 4.1.1.5 and Table 3a):

  High Priority (high): Time-critical messages, such as alarms, synchronization and coordination data.

  Low Priority (low): Less urgent messages, such as process, diagnostic, program data.

This primitive is passed from the Local User to the local FDL controller to send data using Send_Data_with_Acknowledge to a Remote User. Receipt of the primitive results in the transmittal of the L_sdu by the local FDL controller employing the SDA procedure. While processing a SDA Request (i. e. while waiting for the acknowledgement) no further SDA or SDN or SRD shall be sent from the local FDL controller.

FDL_DATA_ACK.indication
(SSAP, DSAP, Loc_add, Rem_add, L_sdu, Serv_class)

- The parameters SSAP and DSAP specify the Source and Destination Service Access Point of the received SDA frame (see Part 4, subclause 4.8.2.2).

- The parameters Loc_add (Local_address) and Rem_add specify the source FDL address (SA, SAE[b7=1]) and the destination FDL address (DA, DAE[b7=1]) of the received SDA frame (see Part 4, subclause 4.8.2). The global address is not permissible in either case.

- The parameter L_sdu contains the Local User data of the received SDA frame.

- The parameter Serv_class specifies the FDL priority of the received SDA frame.

This primitive is passed from the remote FDL controller to the Remote User after receipt of a SDA frame, if the acknowledgement frame was sent. The reaction of the Remote User upon receipt of this primitive is not specified. In case of repetition of the acknowledgement frame, caused by the receipt of a repetition of the SDA frame, the Indication primitive shall not be repeated.

FDL_DATA_ACK.confirm
(SSAP, DSAP, Rem_add, Serv_class, L_status)

- The parameter SSAP specifies the service access point of the Local User. An SSAP-value of 63 is not permissible.

- The parameter DSAP specifies the service access point of the Remote User.

- The parameter Rem_add specifies the destination FDL address (DA, DAE[b7=1]) of the SDA frame. The global address is not permissible.

- The parameter Serv_class specifies the FDL priority of the associated data transfer.

- The parameter L_status indicates success or failure of the preceding SDA request and whether a temporary or a permanent error exists. The following parameter values are specified:

Table 1. SDA, Values of L_status

Code	Meaning	temporary/ permanent
OK	Positive acknowledgement, service finished	-
RR	Negative acknowledgement, resources of the remote FDL controller not available or not sufficient.	t
UE	Negative acknowledgement, remote FDL User/ FDL interface error.	p
RS	Service or Rem_add at remote LSAP or remote LSAP not activated (see subclause 4.2.3.7).	p
LS	Service at local LSAP or local LSAP not activated.	p
LR	Resources of the local FDL controller not available or not sufficient.	t
NA	No or no plausible reaction (Ack./Res.) from remote station.	t
DS	Local FDL/PHY controller not in logical token ring or disconnected from line.	p
IV	Invalid parameters in request.	-

This primitive is passed as an indication from the local FDL controller to the Local User upon completion of the requested SDA service. The reaction of the Local User upon receipt of this primitive is not specified. When L_status indicates a temporary error, the Local User may assume that a subsequent repetition may be successful. In case of a permanent error, management should be consulted prior to repetition of the service. In case of the local errors LS, LR, DS and IV no request frame is transmitted.

4.1.3.2 Send Data with No Acknowledge (SDN)

The Local User prepares a L_sdu for a single, for a group of, or for all Remote Users. The L_sdu is passed to the local FDL controller via the FDL interface by means of a FDL_DATA.request primitive. The FDL controller accepts the service request and tries to send the data to the remote FDL controller requested, to the group, or to all stations (see Part 4, Table 3a, b7=1, Code No 4/6).

The FDL controller returns a local confirmation of transmittal to the Local User by means of a FDL_DATA.confirm primitive.

There is no guarantee of correct receipt at the remote FDL controllers, as neither acknowledgements are given nor local retries take place. Once the data is sent it reaches all Remote Users at the same time (not taking into account signal propagation time). Each addressed remote FDL controller that has received the data error-free passes it to the FDL user by means of a FDL_DATA.indication primitive.

Parameters of the primitives:

FDL_DATA.request
(SSAP, DSAP, Rem_add, L_sdu, Serv_class)

- The parameters have the same meaning as described for SDA under the FDL_DATA_ACK.request primitive. Exception: the global address (Broadcast/ Multicast message) is not permitted for Rem_add. For broadcast messages a DSAP value of 63 shall be chosen. In case of multicast messages the selection (group of stations) is performed by means of a dedicated DSAP.

This primitive is passed from the Local User to the local FDL controller to send data using Send_Data_with_No_ Acknowledge to a single, a group of, or all Remote Users. Receipt of the primitive causes the local FDL controller to transmit the L_sdu by means of the SDN procedure.

FDL_DATA.indication
(SSAP, DSAP, Loc_add, Rem_add, L_sdu, Serv_class)

- The parameters have the same meaning as described for SDA under the FDL_DATA_ACK.indication primitive corresponding to the received SDN frame. Exception: the global address is permitted for Rem_add.

This primitive is passed from the remote FDL controller to the Remote User after receipt of a SDN frame. The reaction of the Remote User upon receipt of this primitive is not specified.

FDL_DATA.confirm
(SSAP, DSAP, Rem_add, Serv_class, L_status)

- The parameters SSAP, DSAP, Rem_add and Serv_class have the same meaning as described for SDA under the FDL_DATA_ACK.confirm primitive corresponding to the SDN frame. Exception: the global address is permitted for Rem_add.

- The parameter L_status indicates local success or failure of the associated SDN request. The following parameter values are specified:

Table 2. SDN, Values of L_status

Code	Meaning	temporary/ permanent
OK	Transmission of data completed by local FDL/PHY controller.	-
LS	Service in local LSAP or local LSAP not activated.	p
LR	Resources of the local FDL controller not available or not sufficient.	t
DS	Local FDL/PHY controller not in logical token ring or disconnected from line.	p
IV	Invalid parameters in request.	-

This primitive is passed from the local FDL controller to the Local User as an indication upon completion of the requested SDN service. The reaction of the Local User upon receipt of this primitive is not specified. When L_status indicates a temporary error, the Local User may assume that a subsequent repetition may be successful. In case of a permanent error, management should be consulted prior to repetition of the service. In case of the local errors LS, LR, DS and IV no request frame is transmitted.

4.1.3.3 Send and Request Data with Reply (SRD)

The Local User prepares a L_sdu for the Remote User. The data is passed to the local FDL controller via the FDL interface by means of a FDL_DATA_ REPLY.request primitive, requesting data from the Remote User at the same time. The FDL controller accepts this combined service request and tries to send the L_sdu together with a data request to the remote FDL controller (see Part 4, Table 3a, b7=1, Code No 12/13). If no data is to be sent at the Local User's end, he may use the service employing the same primitive as an exclusive data request.

After receiving the data and request frame error-free, the remote FDL controller immediately starts transmitting the requested data (see Part 4, Table 3a, b7=0, Code No 8/10) that was previously provided by the Remote User by means of a FDL_REPLY_ UPDATE.request primitive. In case of a FDL_REPLY_ UPDATE without a corresponding SSAP or in case of an error, an acknowledgement is returned (see Part 4, Table 3a, b7=0, Code No 1/2/3/9/12/13). The received L_sdu and the information concerning the back-transmission is passed to the Remote User by means of a FDL_DATA_REPLY. indication primitive via the FDL user interface. If an answer or an acknowledgement is not received within the slot time T_{SL}, the local FDL controller shall repeat the data transmission and request. Between the original data transmission and the requested answer no other traffic takes place on the PROFIBUS. By means of a FDL_DATA_REPLY.confirm primitive the local FDL controller signals to the Local User that the data was duly received by the remote FDL controller and returns the requested data or an error message.

The Remote User is responsible for valid data in the remote FDL controller. By means of a FDL_REPLY_UPDATE.request primitive the Remote FDL User may initiate loading of this data. Upon completion of loading the data area the FDL controller informs the FDL user by means of a FDL_REPLY_UPDATE.confirm primitive.

Parameters of the primitives:

FDL_DATA_REPLY.request
(SSAP, DSAP, Rem_add, L_sdu, Serv_class)

- The parameters SSAP, Rem_add, L_sdu and Serv_class have the same meaning as described for SDA under the FDL_DATA_ACK.request primitive. The L_sdu may have a length of zero.

- The parameter DSAP specifies the service access point for the requested data area and the data to be sent (shared_data_area) as well as the Remote User that receives the indication. A DSAP value of 63 (global access address) is not permissible.

This primitive is passed from the Local User to the local FDL controller to send data to a Remote User and to simultaneously request data from that user or to exclusively request data. Receipt of this primitive by the local FDL controller causes transmittal of the L_sdu with data request by means of the SRD procedure. The L_sdu may have a length of zero.

During the processing of a SRD request (i. e. while waiting for a response or an acknowledgement, respectively) no further service shall be sent from the local FDL controller.

FDL_DATA_REPLY.indication
(SSAP, DSAP, Loc_add, Rem_add, L_sdu, Serv_class, Update_status)

- The parameters have the same meaning as described for SDA under the FDL_DATA_ACK.request primitive with respect to the received SRD frame. The L_sdu may have a length of zero.

- The parameter Update_status specifies, whether or not the response data (L_sdu) has been passed to the local FDL controller. The following parameter values are specified:

Table 3. SRD, Values of Update_status

Code	Meaning	temporary/ permanent
NO	No reply data (L_sdu) transmitted.	t
LO	Low priority reply data transmitted.	-
HI	High priority reply data transmitted.	-

This primitive is passed from the remote FDL controller to the

Remote User when the response/acknowledgement frame for a SRD frame was sent. The reaction of the Remote User upon receipt of the primitive is not specified. In case of a repetition of the response/acknowledgement frame caused by the receipt of a repetition of the SRD frame, the Indication primitive shall not be repeated. The primitive is not applicable when the parameter Indication_mode = Data (see subclause 4.2.3.7) and if the length of the L_sdu is zero both in the received frame and in the corresponding Reply Update (response data). In this case the resources of the LSAP are not used.

FDL_DATA_REPLY.confirm
(SSAP, DSAP, Rem_add, L_sdu, Serv_class, L_status)

- The parameters SSAP, DSAP and Rem_add have the same meaning as described for SDA under the FDL_DATA_ACK.request primitive with respect to the SRD frame. A DSAP value of 63 is not permissible.

- The parameter Serv_class specifies the FDL priority of the request frame.

- The parameter L_sdu contains the requested data of the remote FDL controller.

- The parameter L_status contains the result of the corresponding SRD request and encompasses the values UE, RS, LS, LR, NA, DS and IV as specified for SDA (see Table 1.) and additionally:

Table 4. SRD, Values of L_status

Code	Meaning	temporary/ permanent
DL	Positive acknowledgement for sent data, reply data (L_sdu) with low priority available.	-
DH	Positive acknowledgement for sent data, reply data with high priority available.	-
NR	Positive acknowledgement for sent data, negative acknowledgement for reply data, as not available to the remote FDL controller.	t
RDL	Negative acknowledgement for sent data, resources of the remote FDL controller not available or not sufficient, reply data with low priority available.	t
RDH	Negative acknowledgement for sent data, resources of the remote FDL controller not available or not sufficient, reply data with high priority available.	t
RR	Negative acknowledgement, resources of the remote FDL controller not available or not sufficient and reply data not available.	t

This primitive is passed from the local FDL controller to the Local User to signal success or failure of the previous data transfer and request and to hand over the requested data in case of an error-free procedure. The reaction of the Local User upon receipt of this primitive is not specified. If L_status indicates a temporary error a later repetition may be successful. In case of a permanent error the Local User should consult management prior to repeating the service.

FDL_REPLY_UPDATE.request
(SSAP, L_sdu, Serv_class, Transmit)

- The parameter SSAP specifies the service access point of the Remote User that carries out this request and which data area (shared_data_area) shall be updated by the L_sdu. A SSAP value of 63 is not permissible.

- The parameter L_sdu specifies the contents of the data area that is identified by the SSAP parameter.

- The parameter Serv_class has the same meaning as described for SDA under the FDL_DATA_ACK.request primitive.

- The parameter Transmit specifies whether the Update is transmitted only once (single) or many times (multiple), i. e. in the case of "multiple" the data area is transferred again for each subsequent SRD.

The primitive is passed from the Remote User to the remote FDL controller to receive the response data. Receipt of this primitive causes the remote FDL controller to try to access the data in the course of which data consistency shall be guaranteed. This primitive in connection with the SRD of the Local User is only of importance to the Remote User.

FDL_REPLY_UPDATE.confirm
(SSAP, Serv_class, L_status)

- The parameters SSAP and Serv_class have the same meaning as described for SDA under the FDL_DATA_ACK.confirm primitive.

- The parameter L_status indicates the result of the corresponding UPDATE request. The following parameter values are specified:

Table 5. SRD, Values of L_status (UPDATE)

Code	Meaning	temporary/ permanent
OK	Update data (L_sdu) loaded.	-
LS	Service at local LSAP or local LSAP not activated.	p
LR	Resources of the local FDL controller not available or not sufficient.	t
IV	Invalid parameters in request.	-

This primitive is passed from the remote FDL controller to the Remote User to signal success or failure of the UPDATE request. The reaction of the Remote User upon receipt of this primitive is not specified. If L_status indicates a temporary error, a later repetition may be successful. In case of a permanent error the Local User should consult management prior to repeating the service.

4.1.3.4 Cyclic Send and Request Data with Reply (CSRD)

The Local User prepares a L_sdu each for one, many, or all Remote Users. For each Remote User the data is passed to the local FDL controller by means of a FDL_SEND_UPDATE.request primitive. The addresses and the sequence of the addressed remote stations shall be specified by the Local User by means of the Poll List. The Local User passes the Poll List by means of a FDL_CYC_DATA_REPLY.request primitive where initially all entries are set to "lock". At the same time this action requests the cyclic send including the data request (cyclic low priority SRD). The FDL controller accepts this combined service request and signals the acceptance of the Poll List by means of a FDL_CYC_DATA_REPLY.confirmation.

To allow for a prioritization of the stations to be polled, multiple identical FDL address are permissible in the Poll List.

The FDL controller shall only start the cyclic send of the L_sdu with a data request (see Part 4, Table 3a, b7=1, Code No 12) to the requested remote FDL controller if the corresponding marker is set to "unlock". This shall have been initiated by the Local User by means of a FDL_CYC_ENTRY.request primitive.

If no data is to be sent then a SRD frame without data is sent to each remote station.

After each data transfer and request an immediate response or acknowledgement is expected (see Part 4, Table 3a, b7=0, Code No 8/10 or Code No 1/2/3/9/12/13, respectively). The local and remote FDL controllers react as described for the SRD (see subclause 4.1.3.3). The local FDL controller cyclically passes the requested data or an error message to the Local User by means of a FDL_CYC_DATA_REPLY.confirm primitive.

If the send data is transmitted only once (Transmit = single, cf. FDL_SEND_UPDATE.request), the subsequent SRD frames shall not contain send data until a new update is available.

The Local User may pass new data to the Remote User by means of a FDL_SEND_UPDATE.request primitive at any time.

During the Cyclic Send and Request Data with Reply the Local User shall not request any further cyclic service. The acyclic services SDA, SDN and SRD are permissible. The FDL controller performs the cyclic SRD service with the given Poll List as long as a FDL_CYC_DEACT.request primitive is issued by the Local User.

Parameters of the primitives:

FDL_SEND_UPDATE.request
(SSAP, DSAP, Rem_add, L_sdu, Transmit)

- The parameter SSAP specifies the service access point of the Local User that carries out this request and which data area (shared_data_area) is updated by means of the L_sdu. A SSAP value of 63 is not permissible.

- The parameter DSAP specifies the service access point of the Remote User. A DSAP value of 63 is not permissible.

- The parameter Rem_add contains the destination FDL address of the remote station.

- The parameter L_sdu specifies the contents of the data area that is identified by means of the SSAP parameter. For the L_sdu a length of zero is permissible.

- The parameter Transmit specifies whether the Update is transmitted once (single) or many times (multiple). In the case of "multiple" the data area is transmitted with each further SRD.

The primitive is passed from the Local User to the local FDL controller to receive the send data. The receipt of this primitive causes the local FDL controller to try to receive the data in the course of which data consistency shall be guaranteed.

FDL_SEND_UPDATE.confirm
(SSAP, DSAP, Rem_add, L_status)

- The parameters SSAP, DSAP and Rem_add have the same meaning as described for the FDL_SEND_UPDATE.request primitive.

- The parameter L_status indicates the result of the corresponding UPDATE request. The following parameter values are specified:

Table 6. CSRD, Values of L_status (UPDATE)

Code	Meaning	temporary/ permanent
OK	Send update data (L_sdu) loaded.	-
LS	Service at local LSAP or local LSAP not activated.	p
LR	Resources of the local FDL controller not available or not sufficient or Rem_add/ DSAP not yet in the Poll List.	t/p
IV	Invalid parameters in request.	-

This primitive is passed from the local FDL controller to the Local User to indicate the success or the failure of the UPDATE request. The reaction of the Local User upon receipt of this primitive is not specified.

FDL_CYC_DATA_REPLY.request
(SSAP, Poll_list)

- The parameter SSAP specifies the LSAP of the Local User. A SSAP value of 63 is not permissible.

- The parameter Poll_list may be of the following values:

Table 7. Poll_list

Entry	Name	Meaning	
1	Poll_list_length	Length of Poll List (3 to p+1)	
2	Rem_add	Remote FDL address (DA)	\first
3	DSAP	Destination LSAP (DAE)	/Entry
4	Rem_add	Remote FDL address (DA)	
5	DSAP	Destination LSAP (DAE)	
.	...	...	
.	...	...	
.	...	...	
p	Rem_add	Remote FDL address (DA)	\last
p+1	DSAP	Destination LSAP (DAE)	/Entry

- The parameters Rem_add (0 to 126 [EXT]) and DSAP (0 to 62) have the same meaning as described for SDA under the FDL_DATA_ACK.request primitive. Rem_add as global address and the DSAP value of 63 are not permissible. For remote stations to which no LSAP is transferred, the parameter DSAP is NIL.

This primitive is passed from the Local User to the local FDL controller to cyclically send data to the Remote User and to request data from these users at the same time. The receipt of the primitive causes the local FDL controller to immediately start the cyclic data transfer with the SRD procedure after the marker has been set to "unlock". During the execution of the single cyclic SRD (i. e. while waiting for the response or the acknowledgement) no further service may be transmitted by the local FDL controller.

FDL_CYC_DATA_REPLY.confirm
(SSAP, DSAP, Rem_add, L_sdu, Serv_class, L_status, Update_status)

- The parameters SSAP, DSAP and Rem_add have the same meaning as described for SDA under the FDL_DATA_ACK.request primitive with respect to the SRD frame. A DSAP value of 63 is not permissible.

- The parameters L_sdu, Serv_class and L_status have the same meaning as described for SRD under the FDL_DATA_REPLY.confirm primitive. In case of Poll List delivery the parameter L_status may take the following additional values:

Table 8. CSRD, Values of L_status

Code	Meaning	temporary/ permanent
OK	Poll List accepted.	-
NO	Poll List not accepted, as the Poll List was already available in the local FDL	p

The parameters DSAP, L_sdu and L_status refer to the remote station with the destination FDL address that is specified by the parameter Rem_add.

- The parameter Update_status specifies whether or not the data to be sent (L_sdu) was passed to the remote FDL controller. The following parameter values are specified:

Table 9. CSRD, Values of Update_status

Code	Meaning	temporary/ permanent
NO	No send data (L_sdu) transmitted.	t
LO	Send data with low priority transmitted.	-

This primitive is passed from the local FDL controller to the Local User to indicate success or failure of the Poll List acceptance. In this case only the parameters SSAP and L_status are significant. Furthermore after each cyclic SRD this primitive is passed to the Local User to indicate the success or the failure of the corresponding Send and Request Data and to transfer the received data when no errors occurred. This primitive is omitted after each cyclic SRD, if the parameter Confirm_mode = Data (see subclause 4.2.3.7) and if the L_sdu has length zero both in the received frame as well as in the corresponding Send Update (send data). In this case the resources of the LSAP are not used. The primitive is not omitted, however, if a L_status value of UE, RS, LR, NA, DS or RR occurs.

FDL_CYC_ENTRY.request
(SSAP, DSAP, Rem_add, Marker)

- The parameter SSAP has the same meaning as described for the FDL_CYC_DATA_REPLY.request primitive.

- The parameters DSAP and Rem_add specify the Poll List entry.

- The parameter Marker with the values "unlock" or "lock" specifies whether the entry "Rem_add/DSAP" in the Poll List is to be unlocked or locked.

This primitive is passed from the Local User to the local FDL controller to unlock or lock an entry (Rem_add/DSAP) in the Poll List, i. e. to cyclically send or not to send SRD frames to the corresponding DSAP of the remote FDL controller.

FDL_CYC_ENTRY.confirm
(SSAP, DSAP, Rem_add, L_status)

- The parameter SSAP has the same meaning as described for the FDL_CYC_DATA_REPLY.request primitive.

- The parameters DSAP and Rem_add specify the Poll List entry.

- The parameter L_status indicates the result of the corresponding ENTRY request. The following parameter values are specified:

Table 10. CSRD, Values of L_status (ENTRY)

Code	Meaning	temporary/ permanent
OK	Marker set.	-
LS	Service at local LSAP or local LSAP not activated.	p
NO	Marker not set, as Rem_add/DSAP parameter is not in the Poll List.	p
IV	Invalid parameters in request.	-

This primitive is passed from the local FDL controller to the Local User to indicate the success or failure of the corresponding entry unlock or lock.

FDL_CYC_DEACT.request
(SSAP)

- The parameter SSAP specifies the LSAP of the Local User with the cyclic SRD. A SSAP value of 63 is not permissible.

This primitive is passed from the Local User to the local FDL controller to terminate the cyclic SRD. The receipt of this primitive causes the local FDL controller to stop the cyclic SRD procedures after passing through the Poll List.

FDL_CYC_DEACT.confirm
(SSAP, L_status)

- The parameter SSAP has the same meaning as described for the FDL_CYC_DEACT. request primitive.

- The parameter L_status indicates the result of the corresponding request. The following parameter values are specified:

Table 11. CSRD, Values of L_status (DEACT)

Code	Meaning	temporary/ permanent
OK	Cyclic SRD service correctly finished.	-
LS	Service at local LSAP or local LSAP not activated.	p
IV	Invalid parameters in request.	-

This primitive is passed from the local FDL controller to the Local User after terminating the cyclic SRD. The reaction of the Local User upon receipt of this primitive is not specified.

FDL_DATA_REPLY.indication
(SSAP, DSAP, Loc_add, Rem_add, L_sdu, Serv_class, Update_status)

FDL_REPLY_UPDATE.request
(SSAP, L_sdu, Serv_class, Transmit)

FDL_REPLY_UPDATE.confirm
(SSAP, Serv_class, L_status)

- Description of the primitives and the parameters: cf. SRD.

4.2 FMA1/2 User - FMA1/2 Interface

This clause describes the management services (FMA1/2 = Fieldbus Management for Layers 1 and 2) that are provided to the FMA1/2 User and the associated service primitives and parameters. One distinguishes between mandatory and optional services.

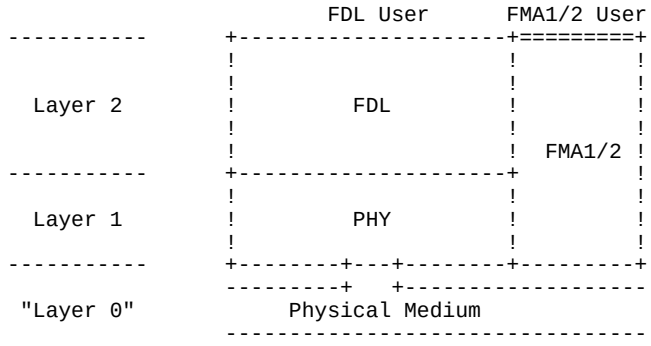


Figure 6. Interface between FMA1/2 User and FMA1/2 in Relation to Layer Model

The service interface between the FMA1/2 user and FMA1/2 provides the following functions:

- Reset of Layers 1 and 2 (local)
- Request for and modification of the actual operating parameters of FDL and PHY (local) and of the counters (local)
- Notification of unexpected events, errors, and status changes (local and remote)
- Request for identification and for the LSAP configuration of the stations (local afnd remote)
- Request for the actual list of stations (Live List) by means of FDL status (remote)
- Activation and deactivation of the local LSAPs

4.2.1 Overview of Services

The FMA1/2 provides the following services to the FMA1/2 user:

- Reset FMA1/2
- Set Value7Read Value FMA1/2
- Event FMA1/2
- Ident FMA1/2
- LSAP Status FMA1/2
- Live List FMA1/2
- (R)SAP Activate / SAP Deactivate FMA1/2

Reset FMA1/2 (mandatory)

The FMA1/2 user employs this service to cause the FMA1/2 to reset Layer 1 (PHY), Layer 2 (FDL) and itself. A reset is equivalent to power on. The FMA1/2 user receives a confirmation thereof.

Set Value FMA1/2 (optional)

The FMA1/2 user employs this service to assign a new value to the variables of Layer 1 or of Layer 2. The user receives a confirmation whether the specified variables have been set to the new value.

Read Value FMA1/2 (optional)

This service enables the FMA1/2 to read variables of Layers 1 and 2. The response of the FMA1/2 contains the actual value of the specified variables.

Event FMA1/2 (mandatory)

The FMA1/2 employs this service to inform the FMA1/2 user about certain events or errors in Layers 1 and 2.

Ident FMA1/2 (optional)

When requesting the identification service, a distinction is made between master and slave stations. By employing this service the FMA1/2 user of a slave station determines the version data of the local FDL and FMA1/2 hard- and software. When employing this service in the case of a master station, the FMA1/2 user may additionally request the same type of information from a remote station.

LSAP Status FMA1/2 (optional)

The FMA1/2 user employs this service to inform himself about the configuration of Service Access Points of the local FDL or, in the case of a remote station (Remote LSAPs), about their FDL services. In the case of a Slave station this service is only possible for local LSAPs.

Live List FMA1/2 (optional)

This service provides the FMA1/2 user of a master station (not possible for slave stations) with an up-to-date list of all stations that are functional on the bus (see Part 4, subclause 4.1.4).

SAP Activate FMA1/2 (optional)

This service enables the FMA1/2 user to activate and to configure a Link Service Access Point (Local LSAP) for the individual FDL services. Excluded from this is the responder function for the reply services (SRD and CSRD). The FMA1/2 user receives a confirmation on the execution of the service from the FMA1/2.

RSAP Activate FMA1/2 (optional)

The FMA1/2 user employs this service to activate a Local LSAP for the responder function of the reply services (SRD and CSRD). The user receives a confirmation of execution from the FMA1/2.

SAP Deactivate FMA1/2 (optional)

The FMA1/2 user employs this service to cause the FMA1/2 to deactivate a Local LSAP. The management returns a confirmation thereof.

4.2.2 Overview of Interactions

The services mentioned in subclause 4.2.1 are described by the following primitives (FMA1/2_...):

Service	Primitive	Possible for the following stations
Reset FMA1/2	FMA1/2_RESET.request	Master and Slave
	FMA1/2_RESET.confirm	- " -
Set Value FMA1/2	FMA1/2_SET_VALUE.request	Master and Slave
	FMA1/2_SET_VALUE.confirm	- " -
Read Value FMA1/2	FMA1/2_READ_VALUE.request	Master and Slave
	FMA1/2_READ_VALUE.confirm	- " -
Service	Primitive	Possible for the following stations
Event FMA1/2	FMA1/2_EVENT.indication	Master and Slave
Ident FMA1/2	FMA1/2_IDENT.request	Local : Master and Slave Remote: Master
	FMA1/2_IDENT.confirm	Local : Master and Slave Remote: Master
LSAP Status FMA1/2	FMA1/2_LSAP_STATUS.request	Local : Master and Slave Remote: Master
	FMA1/2_LSAP_STATUS.confirm	Local : Master and Slave Remote: Master
Live-List FMA1/2	FMA1/2_LIVE_LIST.request	Master
	FMA1/2_LIVE_LIST.confirm	- " -
SAP Activate FMA1/2	FMA1/2_SAP_ACTIVATE.request	Master and Slave
	FMA1/2_SAP_ACTIVATE.confirm	- " -
RSAP Activate FMA1/2	FMA1/2_RSAP_ACTIVATE.request	Master and Slave
	FMA1/2_RSAP_ACTIVATE.confirm	- " -
SAP Deactivate FMA1/2	FMA1/2_SAP_DEACTIVATE.request	Master and Slave
	FMA1/2_SAP_DEACTIVATE.confirm	- " -

Temporal Relationships of Service Primitives:

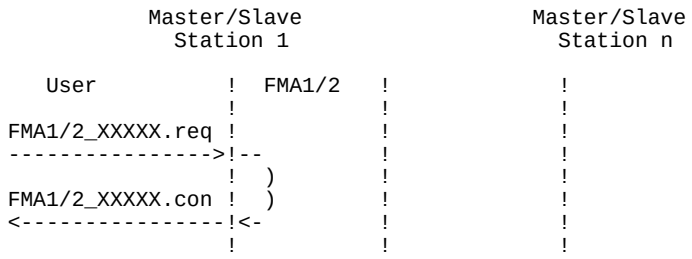


Figure 7. Reset, Set Value, Read Value, Ident(Local), LSAP Status(Local), (R)SAP Activate, SAP Deactivate FMA1/2 Service

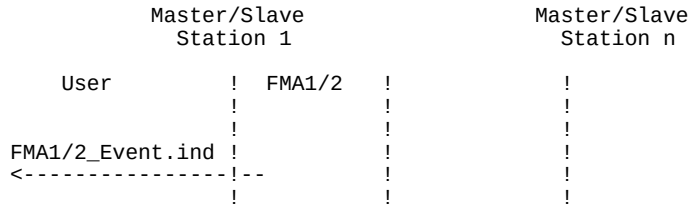


Figure 8. Event FMA1/2 Service

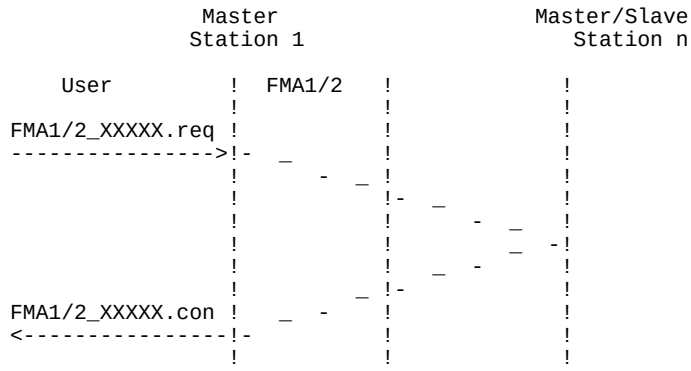


Figure 9. Ident(Remote), LSAP Status(Remote), Live List FMA1/2 Service

4.2.3 Detailed Specification of Services and Interactions

4.2.3.1 Reset FMA1/2

The service "Reset FMA1/2" is **mandatory**.

The FMA1/2 user passes a FMA1/2_RESET.request primitive to the FMA1/2 causing it to reset Layers 1 and 2. As a result the FMA1/2 passes a FDL_RESET.request primitive to Layer 2 (FDL) and a PHY_RESET.request primitive to Layer 1 (PHY) (see subclauses 5.2.3.1 and 5.3.3.1). After the confirmations of FDL and PHY by means of the respective confirmation primitives FMA1/2 resets itself and passes a FMA1/2_RESET.confirm primitive to the user.

Parameter of the primitives

FMA1/2_RESET.request
(MSAP_0)

- The parameter MSAP_0 (Management Service Access Point_0) specifies the service access point of the Local FMA1/2 User. This MSAP is activated and configured depending on the implementation.

FMA1/2_RESET.confirm
(MSAP_0, M_status)

- The parameter MSAP_0 has the same meaning as described for the FMA1/2_RESET. request primitive.

- The parameter M_status contains a confirmation about the execution of the service. The following values are specified:

Table 12. Reset, Values of M_status

Code	Meaning	temporary/ permanent
OK	The Reset function was carried out successfully.	-
NO	The Reset function was not carried out successfully.	t/p
IV	Invalid parameters in request.	-

4.2.3.2 Set Value FMA1/2, Read Value FMA1/2

Set Value FMA1/2

The service "Set Value FMA1/2" is **optional**.

The FMA1/2 user passes a FMA1/2_SET_ VALUE. request primitive to the FMA1/2 to assign a desired value to one or more specified variables of Layers 1 and/or 2. The management passes the corresponding FDL_SET_VALUE.request and/or PHY_SET_ VALUE.request primitives to the respective Layers and passes a FMA1/2_SET_VALUE. confirm primitive to the user after receiving all the corresponding confirmation primitives.

Parameters of the primitives

FMA1/2_SET_VALUE.request
(MSAP_0, Variable_name 1 to z, Desired_value 1 to z)

- The parameter MSAP_0 has the same meaning as described for the FMA1/2_RESET. request primitive.

- The parameter Variable_name contains a list of one or more variables of the FDL and/or PHY. The selectable variables are specified in subclauses 5.2.3.2 and 5.3.3.2.

- The parameter Desired_value contains a list of one or more values for the specified variables. The permissible values or ranges of values are specified in subclauses 5.2.3.2 and 5.3.3.2.

FMA1/2_SET_VALUE.confirm
(MSAP_0, M_status 1 to z)

- The parameter MSAP_0 has the same meaning as described for the FMA1/2_RESET. request primitive.

- The parameter M_status contains a confirmation about the execution of the service. The following values are specified:

Table 13. Set Value, Values of M_status

Code	Meaning	temporary/ permanent
OK	The Variable has the new value.	-
NO	The Variable does not exist or could not be set to the new value.	t/p
IV	Invalid parameters in request.	-

Read Value FMA1/2

The service "Read Value FMA1/2" is **optional**.

The FMA1/2 user passes a FMA1/2_READ_ VALUE.request primitive to the FMA1/2 to read the current value of one or more variables of the FDL or PHY, respectively. After receipt of this primitive the FMA1/2 creates the corresponding FDL_READ_ VALUE.request and/or PHY_READ_VALUE.request primitives and passes a FMA1/2_READ_ VALUE.confirm primitive to the FMA1/2 user after receipt of all corresponding confirmation primitives. This primitive contains as a parameter one or more of the requested variable values.

Parameters of the primitives

FMA1/2_READ_VALUE.request
(MSAP_0, Variable_name 1 to z)

- The parameter MSAP_0 has the same meaning as described for the FMA1/2_RESET. request primitive.

- The parameter Variable_name specifies one or more variables whose values shall be read. The variables that may be chosen are specified in subclauses 5.2.3.2 and 5.3.3.2.

FMA1/2_READ_VALUE.confirm
(MSAP_0, Current_value 1 to z, M_status 1 to z)

- The parameter MSAP_0 has the same meaning as described for the FMA1/2_RESET. request primitive.

- The parameter Current_value contains the actual value of the variables that were specified in the preceding FMA1/2_READ_VALUE.request. The permissible values or ranges of values are specified in subclauses 5.2.3.2 and 5.3.3.2.

- The parameter M_status contains a confirmation about the execution of the service. The following values are specified:

Table 14. Read Value, Values of M_status

Code	Meaning	temporary/ permanent
OK	The Variable could be read.	-
NO	The Variable does not exist or could not be read. The corresponding value of Current_value is not defined.	t/p
IV	Invalid parameters in request.	-

4.2.3.3 Event FMA1/2

The service "Event FMA1/2" is **mandatory**.

After receipt of a PHY_EVENT.indication or FDL_Fault.indication primitive the FMA1/2 passes a FMA1/2_EVENT.indication primitive to the FMA1/2 user to inform it about important events or error conditions in the respective Layers.

Parameters of the primitives

FMA1/2_EVENT.indication
(MSAP_1, Event/Fault, Add_info)

- The parameter MSAP_1 specifies the service access point of the local FMA1/2 user. This MSAP is activated and configured depending on the implementation.

- The parameter Event/Fault specifies the event that took place or the reason for the error and may take the values of the parameter "Variable_name" (see subclause 5.3.3.3) or "Fault_type" (see subclause 5.2.3.3), respectively.

- The parameter Add_info contains additional information concerning the respective event and may take the values of the parameter New_value of the PHY_EVENT.indication primitive (see subclause 5.3.3.3).

4.2.3.4 Ident FMA1/2

The service "Ident FMA1/2" is **optional**.

By means of a FMA1/2_IDENT.request primitive the FMA1/2 user requests the FMA1/2 to carry out a station identification. The FMA1/2 then passes a FDL_IDENT. request primitive to the FDL. If the user requests the identification of a remote station, the FDL Layer shall send a corresponding request to this station by means of a Request Ident with Reply (see Part 4, Table 3a, b7=1, Code No 14) and return the result to the FMA1/2 by means of a FDL_IDENT.confirm primitive (see Part 4, Table 3a, b7=0, Code No 1/8/9). If the identification refers to the local FDL Layer, the FDL shall immediately reply with a FDL_ IDENT.confirm primitive. After receipt of a FDL_IDENT.confirm primitive the FMA1/2 returns the requested data to the user with a FMA1/2_IDENT.confirm primitive.

Parameters of the primitives

FMA1/2_IDENT.request
(MSAP_0, Rem/Loc_add)

- The parameter MSAP_0 has the same meaning as described for the FMA1/2_RESET. request primitive. In the case of remote requests the value of the parameters is not transferred in the frame.

- The parameter Rem/Loc_add contains in the case of a remote request the FDL address of the remote station. The global address is not permissible. In the case of local requests the parameter contains the own FDL address (TS).

FMA1/2_IDENT.confirm
(MSAP_0, Rem/Loc_add, Ident_list, M_status)

- The parameter MSAP_0 has the same meaning as described for the FMA1/2_RESET. request primitive.

- The parameter Rem/Loc_add has the same meaning as described for the FMA1/2_ IDENT.request primitive.

- The parameter Ident_list specifies a list of values for the identification of a station:

  Vendor_name : Name of the vendor
  Controller_type : Type of the PROFIBUS interface
  HW/SW_release : Version number of the hardware and software

- The parameter M_status contains a confirmation concerning the execution of the service. The following values are specified:

Table 15. Ident, Values of M_status

Code	Meaning	temporary/ permanent
OK	The Identification could be carried out.	-
LR	Resources of the local FDL controller not available or not sufficient.	t
NA	No or no plausible reaction (Ack./Res.) from remote station.	t
DS	Local FDL/PHY controller not in logical token ring or disconnected from line.	p
NR	Negative acknowledgement for Ident data, as not available to the remote FDL controller.	p
IV	Invalid parameters in request.	-

4.2.3.5 LSAP Status FMA1/2

The service "LSAP Status FMA1/2" is **optional**.

The FMA1/2 user passes a FMA1/2_LSAP_STATUS. request primitive to the FMA1/2 to request the configuration of a LSAP with respect to the FDL services. The FMA1/2 then generates a corresponding FDL_LSAP_ STATUS.request primitive for the FDL. If the user requests the LSAP status of a remote station, the FDL Layer issues a corresponding request by means of a Request LSAP Status with Reply (see Part 4, Table 3a, b7=1, Code No 15) to this station and passes the response (see Table 3a, b7=0, Code No 1/3/8/9) by means of a FDL_LSAP_STATUS.confirm primitive to the FMA1/2. If the LSAP status refers to the local FDL Layer, the FDL immediately responds by means of a FDL_LSAP_STATUS.confirm primitive. After receipt of a FDL_LSAP_STATUS.confirm primitive, the FMA1/2 passes the parameter values to the FMA1/2 user by means of a FMA1/2_LSAP_STATUS.confirm primitive.

Parameters of the primitives

FMA1/2_LSAP_STATUS.request
(MSAP_0, LSAP, Rem/Loc_add)

- The parameter MSAP_0 has the same meaning as described for the FMA1/2_RESET. request primitive. In the case of remote requests the value of the parameter is not transferred in the frame.

- The parameter LSAP specifies the remote/local LSAP whose configuration is requested. The LSAP values 0 to 63 and NIL are permitted. If the configuration of the default LSAP is to be determined, the parameter LSAP shall have

the value NIL. In this case no DAE with b7=0 exists in the request frame for a remote request.

- The parameter Rem/Loc_add contains the FDL address of the remote station in the case of a remote request. The global address is not permissible. In the case of a local request the parameter contains the own FDL address (TS).

FMA1/2_LSAP_STATUS.confirm
(MSAP_0, LSAP, Rem/Loc_add, Access, Service_type 1 to z, Role_in_service 1 to z, M_status)

- The parameter MSAP_0 has the same meaning as described for the FMA1/2_RESET. request primitive.

- The parameters LSAP and Rem/Loc_add have the same meaning as described for the FMA1/2_LSAP_STATUS.request primitive.

- The parameter Access with the values All/0 to 126 [EXT] specifies the access protection. "All" means that all remote stations have access to this LSAP. Apart from that only the specified remote station (address 0 to 126 and if applicable region/segment address [EXT]) has access.

- The parameter Service_type contains the FDL services that are activated at the remote/local LSAP. The following values are specified:

  SDA, SDN, SRD and CSRD

- The parameter Role_in_service specifies the configuration for the activated service. The following values are specified:

  Initiator: The station initiates the respective service exclusively.

  Responder: The station responds to the service exclusively.

  Both: The station initiates and responds to the service.

- The parameter M_status contains a confirmation about the execution of the service. The following values are specified:

Table 16. LSAP Status, Values of M_status

Code	Meaning	temporary/ permanent
OK	The LSAP Status could be read.	-
LR	Resources of the local FDL controller not available or not sufficient.	t
LS	The local LSAP is not activated.	d
RS	The LSAP is not activated in the remote FDL controller.	p
NA	No or no plausible reaction (Ack./Res.) from remote station.	t
DS	Local FDL/PHY controller not in logical token ring or disconnected from line.	p
NR	Negative acknowledgement for LSAP data, as not available in the remote FDL controller.	p
IV	Invalid parameters in request.	-

4.2.3.6 Live List FMA1/2

The service "Live List FMA1/2" is **optional**.

The FMA1/2 user requests by means of a FMA1/2_ LIVE_LIST.request primitive an up-to-date list of all stations that may be currently reached on the bus. The FMA1/2 passes this request to the FDL with a FDL_LIVE_LIST.request primitive. After this request the FDL Layer performs a Request FDL Status with Reply for every possible station address (see Part 4, subclause 4.1.4, Table 3a, b7=1, Code No 9) and passes the result of this survey (see Part 4, Table 3a, b7=0, Code No 0) to the FMA1/2 with a FDL_LIVE_LIST.confirm primitive. The FMA1/2 passes this list to the user with a FMA1/2_LIVE_LIST.confirm primitive.

Parameters of the primitives

FMA1/2_LIVE_LIST.request
(MSAP_0)

- The parameter MSAP_0 has the same meaning as described for the FMA1/2_RESET. request primitive. The value of the parameter is not transferred in the request frames.

FMA1/2_LIVE_LIST.confirm
(MSAP_0, Live_list, M_status)

- The parameter MSAP_0 has the same meaning as described for the FMA1/2_RESET. request primitive.

- The parameter Live_list corresponds to the Live List described in Part 4, subclause 4.1.4.

- The parameter M_status contains a confirmation about the execution of the service. The following values are specified:

Table 17. Live List, Values of M_status

Code	Meaning	temporary/ permanent
OK	The live list is complete.	-
LR	Resources of the local FDL controller not available or not sufficient.	t
DS	Local FDL/PHY controller not in logical token ring or disconnected from line.	p
IV	Invalid parameters in request.	-

4.2.3.7 (R)SAP Activate FMA1/2, SAP Deactivate FMA1/2

SAP Activate FMA1/2

The service "SAP Activate FMA1/2" is **optional**.

This service provides the FMA1/2 user with the possibility to activate and to configure a local LSAP for the individual FDL services. An exception to this is the responder function for the Reply services (SRD, CSRD) that are activated by means of the RSAP Activate service.
After receipt of the FMA1/2_SAP_ACTIVATE.request primitive from the user the FMA1/2 passes a FDL_SAP_ACTIVATE.request primitive to the FDL. After receipt of

the corresponding Confirmation primitive the FMA1/2 passes a FMA1/2_SAP_ACTIVATE.confirm primitive to the user.

Parameters of the primitives

FMA1/2_SAP_ACTIVATE.request
(MSAP_2, SSAP, Access, Service_list)

- The parameter MSAP_2 specifies the service access point of the local FMA1/2 User. This MSAP is activated and configured depending on the implementation.

- The parameter SSAP specifies the local LSAP that is to be activated and configured. The SSAP values 0 to 63 and NIL are permissible.

- The parameter Access with the values All/0 to 126 [EXT] is used for access protection and specifies whether all remote stations (All) or only an individual remote station (0 to 126 and if applicable Region/Segment address [EXT]) may have access to the LSAP (Send/Request Data). The parameter is only valid for the responder function(s), i. e. Role_in_service = Responder/Both.

- The parameter Service_list is a list containing further parameters:

Table 18. SAP Activate Service_list

Entry	Name	
1	Service_list_length	(1 + 3·z)
2	Service_activate	1
3	Role_in_service	1
4	L_sdu_length_list	1
5	Service_activate	2
6	Role_in_service	2
7	L_sdu_length_list	2
.	. . .	
.	. . .	
.	. . .	
k	Service_activate	z
k+1	Role_in_service	z
k+2	L_sdu_length_list	z
	z = maximum 4	

- The parameter Service_activate contains the FDL service that shall be activated for this Access Point (SDA, SDN, SRD or CSRD).

- The parameter Role_in_service specifies the configuration for the service to be activated. The following values are specified:

  Initiator: The station initiates the respective service exclusively.

  Responder: The station responds to the service exclusively. Not permissible for SRD and CSRD.

  Both: The station initiates and responds to the service. Not permissible for SRD and CSRD.

- The parameter L_sdu_length_list is a list of Max_L_sdu lengths. For the services SDA, SDN and SRD the list has the following fixed structure:

Table 19. SAP Activate L_sdu_length_list

Entry	Name
1	Max_L_sdu_length_req_low
2	Max_L_sdu_length_req_high
3	Max_L_sdu_length_ind/con_low
4	Max_L_sdu_length_ind/con_high

- The parameter Max_L_sdu_length_req_low or _high specifies the maximum length of the low or high priority L_sdu to be sent that is passed to the initiator by means of the Request primitive for the services SDA, SDN, and SRD. For the Indication of the services SDA and SDN at the Responder and for the Confirmation of the service SRD at the Initiator, the maximum length of the L_sdu to be received is specified by the parameter Max_L_sdu_length_ind/con_low or _high. The length of the L_sdu may be 0 to 246 octets. When using SSAP, DSAP and Region/Segment addresses a maximum of 242 octets is permissible (see Part 4, subclause 4.7.5).

  Depending on the service and Role_in_service the following lengths > 0 octets ("x": > 0; "-": = 0) are permissible:

Table 20. SAP Activate, L_sdu Lengths for SDA and SDN

Service: SDA and SDN														
Length	Initiator		Responder				Both							
1	x	-	x	-	-	-	x	-	x	-	x	-	x	x	x
2	-	x	x	-	-	-	-	x	-	x	-	x	x	x	x
3	-	-	-	x	-	x	x	-	-	x	x	x	x	-	x
4	-	-	-	-	x	x	-	x	x	-	x	x	-	x	x

1 to 4 denote the lengths as in Table 19

Table 21. SAP Activate L_sdu Lengths of SRD

Service: SRD														
Length	Initiator*)													
1	-	-	-	x	-	-	x	x	x	x	-	x		
2	-	-	-	-	x	x	-	x	x	-	x	x		
3	x	-	x	x	-	x	-	x	-	x	x	x		
4	-	x	x	-	x	-	x	-	x	x	x	x		

1 to 4 denote the lengths as in Table 19.

*) Responder only with RSAP Activate, Both not allowed

For the cyclic service CSRD the L_sdu_length_list has the following structure:

Table 22. SAP Activate L_sdu_length_list

Entry	Name
1	List_length (4 to i)
2	Confirm_mode
3	Poll_list_length (see subclause 4.1.3.4)
4	Rem_add (0 to 126 [EXT])/DSAP (0 to 62):
	a) Max_L_sdu_length_con_low
	b) Max_L_sdu_length_con_high
	c) Max_L_sdu_length_req_low
5	Rem_add (0 to 126 [EXT])/DSAP (0 to 62):
	a) Max_L_sdu_length_con_low
	b) Max_L_sdu_length_con_high
	c) Max_L_sdu_length_req_low
.	...
.	...
.	...
i	Rem_add (0 to 126 [EXT])/DSAP (0 to 62):
	a) Max_L_sdu_length_con_low
	b) Max_L_sdu_length_con_high
	c) Max_L_sdu_length_req_low

- The parameter Confirm_mode with the values All/Data specifies whether for the CSRD service the FDL_CYC_DATA_REPLY.confirm primitive is generated after every cyclic SRD (All) or whether it is omitted (Data) only if both in the received frame (confirmation) and in the corresponding Send Update (data to be sent) L_sdu = Null and no errors occurred (see subclause 4.1.3.4, FDL_CYC_DATA_REPLY.confirmation).

- For each individual Poll List entry (specified by Rem_add/DSAP) up to three L_sdu lengths are permissible. For the confirmation of the CSRD service at the initiator, the maximum length of the L_sdu to be received is specified by the parameter Max_L_sdu_length_con_low or _high.

  The parameter Max_L_sdu_length_req_low specifies the maximum length of the low priority L_sdu to be sent, which is passed to the initiator by means of the Send Update Request primitive of the CSRD service. The length of the L_sdu may be 0 to 246 octets. When using SSAP, DSAP and Region/Segment addresses a maximum of 242 octets is permitted (see Part 4, subclause 4.7.5).

FMA1/2_SAP_ACTIVATE.confirm
(MSAP_2, SSAP, M_status)

- The parameter MSAP_2 has the same meaning as described for the FMA1/2_SAP_ACTIVATE.request primitive.

- The parameter SSAP has the same meaning as described for the FMA1/2_SAP_ACTIVATE.request primitive.

- The parameter M_status contains a confirmation about the execution of the service. The following values are specified:

Table 23. SAP Activate, Values of M_status

Code	Meaning	temporary/ permanent
OK	The SSAP is activated as requested.	-
NO	The SSAP is not activated (already activated or resources not available or not sufficient).	t/p
IV	Invalid parameters in request.	-

RSAP Activate FMA1/2

The service "RSAP Activate FMA1/2" is **optional**.

The FMA1/2 user passes a FMA1/2_RSAP_ACTIVATE. request primitive to the FMA1/2 to activate and to configure a local LSAP for the responder function of the Reply services (SRD and CSRD). The FMA1/2 passes a corresponding FDL_RSAP_ ACTIVATE.request primitive to the FDL and passes a FMA1/2_RSAP_ACTIVATE.confirm primitive to the user after receipt of the corresponding Confirmation primitive.

Parameters of the primitives

FMA1/2_RSAP_ACTIVATE.request
(MSAP_2, SSAP, Access, L_sdu_length_list, Indication_mode)

- The parameter MSAP_2 has the same meaning as described for the FMA1/2_SAP_ ACTIVATE.request primitive.

- The parameter SSAP specifies the local LSAP for the data area (shared_data_ area) and the user that receives the Indication primitive if a SRD is received on this "DSAP". The SSAP values 0 to 62 and NIL are permitted.

- The parameter Access has the same meaning as described for the FMA1/2_SAP_ ACTIVATE.request primitive.

- The parameter L_sdu_length_list is a list of Max_L_sdu length. The list has the following structure:

Table 24. RSAP Activate, L_sdu_length_list

Entry	Name
1	Max_L_sdu_length_req_low
2	Max_L_sdu_length_req_high
3	Max_L_sdu_length_ind_low
4	Max_L_sdu_length_ind_high

- The parameter Max_L_sdu_length_req_low or _high specifies the maximum length of the low or high priority L_sdu that is to be sent in the SRD response frame, which is passed with the Reply Update Request primitive at the responder. For the indication at the responder the maximum length of the L_sdu to be received is specified by the parameter Max_L_sdu_length_ind_low or _high. The length of the L_sdu may be 0 to 246 octets. When using SSAP, DSAP, and Region/Segment addresses a maximum of 242 octets is permissible (see Part 4, subclause 4.7.5).

The following lengths > 0 octets ("x": > 0; "-": = 0) are permissible at the responder:

Table 25. RSAP Activate, L_sdu Length of SRD and CSRD

Service: SRD or CSRD												
Length	Responder											
1	x	-	x	x	-	-	x	x	x	x	-	x
2	-	x	x	-	x	x	-	x	x	-	x	x
3	-	-	-	x	-	x	-	x	-	x	x	x
4	-	-	-	-	x	-	x	-	x	x	x	x
1 to 4 denote the lengths as in Table 24.												

- The parameter Indication_mode with the values All/Data/Unchanged specifies whether in the case of the SRD/CSRD service the FDL_DATA_REPLY. indication primitive shall be generated (All) or whether it shall be omitted (Data) only if both in the received frame (request) and in the corresponding Reply Update (response data) L_sdu = Null.

  The overwriting of a local RSAP (changing of the parameter Access) is informed by the FDL by the parameter Indication_mode contains the value "Unchanged". In this case only the parameter "Access" is overwritten and all other parameters are unchanged.

FMA1/2_RSAP_ACTIVATE.confirm
(MSAP_2, SSAP, M_status)

- The parameter MSAP_2 has the same meaning as described for the FMA1/2_SAP_ ACTIVATE.request primitive.

- The parameter SSAP has the same meaning as described for the FMA1/2_RSAP_ ACTIVATE.request primitive.

- The parameter M_status contains a confirmation about the execution of the service. The following values are specified:

Table 26. RSAP Activate, Values of M_status

Code	Meaning	temporary/ permanent
OK	The SSAP is activated resp. overwritten as requested.	-
NO	Indication_mode All/Data: The SSAP is not activated (already activated or resources not available or not sufficient).	t/p
	Indication_mode Unchanged: The SSAP is not overwritten, as it was not activated.	t/p
IV	Invalid parameters in request.	-

SAP Deactivate FMA1/2

The service "SAP Deactivate FMA1/2" is **optional**.

The FMA1/2 user employs this service to deactivate all FDL services for a local LSAP. After receipt of a FMA1/2_SAP_DEACTIVATE.request primitive from the user, the FMA1/2 passes a corresponding FDL_SAP_DEACTIVATE.request primitive to the FDL controller. The FDL shall test whether a response is still pending and shall deactivate all services for the specified LSAP either directly (if no response is pending) or after receipt of the response. Immediately after this the FDL passes a FDL_SAP_DEACTIVATE.confirm primitive to the FMA1/2. After receipt of the Confirmation primitive the FMA1/2 passes a FMA1/2_SAP_DEACTIVATE.confirm primitive to the user.

Parameters of the primitives

FMA1/2_SAP_DEACTIVATE.request
(MSAP_2, SSAP)

- The parameter MSAP_2 has the same meaning as described for the FMA1/2_SAP_ ACTIVATE.request primitive.

- The parameter SSAP contains the Local LSAP that is to be deactivated for all FDL services.

FMA1/2_SAP_DEACTIVATE.confirm
(MSAP_2, SSAP, M_status)

- The parameter MSAP_2 has the same meaning as described for the FMA1/2_SAP_ ACTIVATE.request primitive.

- The parameter SSAP has the same meaning as described for the FMA1/2_SAP_ DEACTIVATE.request primitive.

- The parameter M_status contains a confirmation about the execution of the service. The following values are specified:

Table 27. SAP Deactivate, Values of M_status

Code	Meaning	temporary/ permanent
OK	The SSAP is deactivated.	-
NO	The SSAP has not been activated.	p
IV	Invalid parameters in request.	-

5 Management (FMA1/2)

The management for the Layers 1 and 2 (FMA1/2) organizes the initialization, the supervision and the error handling between the FMA1/2 User and the logical functions in PHY and FDL.

The PHY/FDL services are either **mandatory** or **optional**.

The relations between FMA1/2 and the Layer 7 (APP and FMA7 respectively) are described in Part 2 of the specification.

5.1 General Description of FMA1/2 Functions

Local FMA1/2 Functions for the Layers 1 and 2:

- reset of the Layers 1 and 2
- reading and setting of parameters
- activation, configuration and deactivation of LSAPs
- event and error messages
- version identification
- determination of LSAP configuration

Remote FMA1/2 Functions for the Layers 1 and 2:

- version identification
- request of LSAP configuration
- determination of the Live List

The FMA1/2 acts as a mediator between the local FMA1/2 user and the Layers 1 and 2. Requests from the FMA1/2 user, modified as required, are transferred to the FDL and PHY control respectively by the FMA1/2, and acknowledged with a confirmation to the local FMA1/2 User. Moreover the FMA1/2 immediately receives indications from the layers if status changes have occurred within the layers. Such messages lead to a FMA1/2 user indication.

5.2 FDL - FMA1/2 Interface

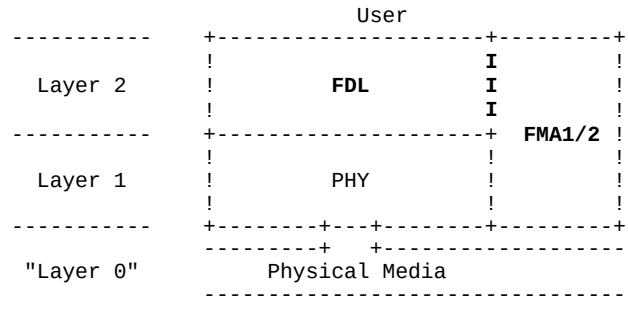


Figure 10. Interface between FDL and FMA1/2 in Relation to Layer Model

5.2.1 Overview of Services

The FDL provides the following services to FMA1/2:

- Reset FDL
- Set Value FDL
- Read Value FDL
- Fault FDL
- Ident FDL
- LSAP Status FDL
- Live List FDL
- SAP Activate FDL
- RSAP Activate FDL
- SAP Deactivate FDL

> identical to the homonymous service without suffix "FDL" in subclause 4.2.1 with the exception, that the parameters MSAP_0 to _2 and M_status "IV" are absent.

Reset FDL (mandatory)

The FMA1/2 uses this service to reset the FDL. After the execution the FMA1/2 receives a confirmation.

Set Value FDL (mandatory)

The Set Value FDL service allows the FMA1/2 to set certain parameters of the FDL to desired values. A confirmation informs the FMA1/2 whether the given parameter could be set to the value or not.

Read Value FDL (optional)

With this service the FMA1/2 is able to read certain FDL parameters. The current value of the given parameter is transferred to the FMA1/2 in the response of the FDL.

Fault FDL (mandatory)

The FDL uses this service to inform the FMA1/2 of faulty conditions and events.

5.2.2 Overview of Interactions

The services mentioned in subclause 4.2.1 are described by the following primitives (FDL_...):

Service	Primitive	permissible for following users
Reset FDL	FDL_RESET.request	Master and Slave
	FDL_RESET.confirm	- " -
Set Value FDL	FDL_SET_VALUE.request	Master and Slave
	FDL_SET_VALUE.confirm	- " -
Read Value FDL	FDL_READ_VALUE.request	Master and Slave
	FDL_READ_VALUE.confirm	- " -
Fault FDL	FDL_FAULT.indication	Master and Slave

Temporal Relationships of Service Primitives:

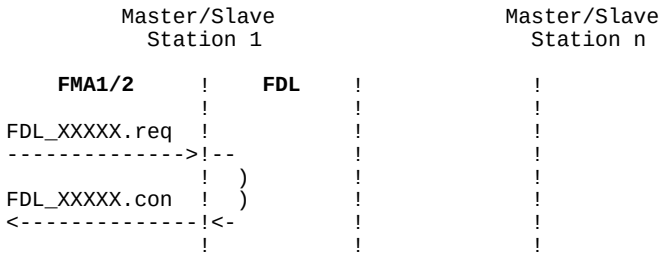


Figure 11. Reset FDL, Set Value FDL, Read Value FDL Service

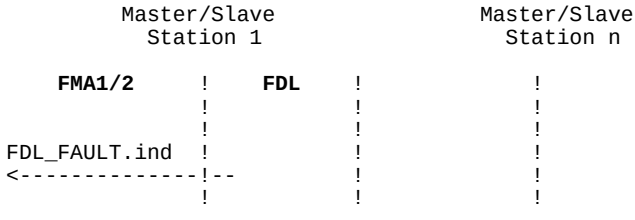


Figure 12. Fault FDL Service

5.2.3 Detailed Specification of Services and Interactions

5.2.3.1 Reset FDL

The service "Reset FDL" is **mandatory**.

With a FDL_RESET.request primitive the FMA1/2 causes a reset of the FDL Layer. This is carried out in the same manner as at a Power On ("Offline"-status; FDL variables [operational parameters/counters] are cleared). Immediately afterwards a FDL_RESET. confirm primitive is transferred to the FMA1/2 .

Parameters of the Primitives

FDL_RESET.request

FDL_RESET.confirm

- Both primitives have no parameters

5.2.3.2 Set Value FDL, Read Value FDL

Set Value FDL

The service "Set Value FDL" is **mandatory**.

The FMA1/2 gives a FDL_SET_VALUE.request primitive to the FDL to set a specified FDL variable to a desired value. After receiving this primitive the FDL tries to select this variable and to deliver the new value. Subsequently the FMA1/2 receives a FDL_SET_ VALUE.confirm primitive.

Parameters of the Primitives

FDL_SET_VALUE.request
(Variable_name, Desired_value)

- The parameter Variable_name specifies the FDL variable to which a new value is to be assigned. The variables shown in Table 28 may be selected.

- The parameter Desired_value specifies the new value for the FDL variable. The selected variable may take the values shown in Table 29.

FDL_SET_VALUE.confirm

- This primitive has no parameters.

Table 28. FDL Variables

Operating Parameters (mandatory)		
Name		Meaning
TS		FDL Address of this station
Baud_rate		Data Signalling Rate of this PROFIBUS System
Medium_red		Availability of Redundant Media
HW-Release		Hardware Release Number
SW-Release		Software Release Number
T_{SL}		Slot Time
min T_{SDR}		smallest Station Delay Time
max T_{SDR}	*)	largest Station Delay Time
T_{QUI}	*)	Transmitter fall Time (Line State Uncertain Time)/Repeater switch Time
T_{SET}	*)	Setup Time
T_{TR}	*)	Target Rotation Time
G	*)	GAP Update Factor
in_ring_desired	*)	Request entry into or exit out of the logical Token Ring
HSA	*)	Highest Station Address on this PROFIBUS System
max_retry limit	*)	Maximum Number of Retries
Counters (optional)		
Name		Meaning
Frame_sent_count	*)	Number of frames sent
Retry_count	*)	Number of frame repeats
SD_count		Number of correct Start Delimiters
SD_error_count		Number of defective Start Delimiters

If a counter reaches its maximum value, then both this counter and its comparison counter are stopped. If a counter is reset to zero the related coparative counter is also reset to zero, then this counters are free to count again.

*) only possible in Master Stations

Table 29. Values of FDL Variables

Operating Parameters (mandatory)	
Variable	Range of Values
TS	1 Octet Address field, Station Address value 0 to 126 and optional Address Extension (see Part 4, subclause 4.7.2)
Baud_rate	9,6; 19,2; 93,75; 187,5; 500; 1500 kbit/s
Medium_red	single; redundant
HW-Release	LE_HR; HW_release (see Part 4, Data Field)
SW-Release	LE_SR; SW_release (see Part 4, Data Field)
T_{SL}	52 to 2^{16}-1 (Bit Times)
min T_{SDR}	2^{0} to 2^{16}-1 (Bit Times)
max T_{SDR}	2^{0} to 2^{16}-1 (Bit Times)
T_{QUI}	0 to 2^{8}-1 (Bit Times)
T_{SET}	2^{0} to 2^{8}-1 (Bit Times)
T_{TR}	2^{0} to 2^{24}-1 (Bit Times)
G	1 to 100
in_ring_ desired	true; false
HSA	2 to 126
max_retry_ limit	1 to 8 (preferably 1)
Counters (optional)	
Variable	Value
Frame_sent_count	0
Retry_count	0
SD_count	0
SD_error_count	0

Read Value FDL

The service "Read Value FDL" is **optional**.

The FMA1/2 delivers a FDL_READ_VALUE.request primitive to the FDL to read the current value of a FDL variable. After receiving this primitive the FDL tries to select the specified variable and to deliver its current value to the FMA1/2 with a FDL_READ_VALUE.confirm primitive.

Parameters of the Primitives

FDL_READ_VALUE.request
(Variable_name)

- The parameter Variable_name specifies the FDL variable whose current value shall be read. In addition to the variables described for the FDL_SET_ VALUE.request primitive (see Table 28) the following may be selected:

Table 30. Additional FDL Variables

Name		Meaning
T_{RR}	*)	Real Rotation Time
LAS	*)	List of Master Stations in the logical ring
GAPL	*)	List all of Stations in the own GAP
	*)	only possible in Master Stations

FDL_READ_VALUE.confirm
(Current_value)

- The parameter Current_value contains the current value of the variable which was specified in the preceding FDL_READ_VALUE.request primitive. In addition to the ranges of values defined for the FDL_SET_VALUE.request primitive (see Table 29) the following ranges are permissible for the variables and the counters above:

Table 31. Values of the additional FDL Variables

Variable	Range
T_{RR}	2^0 to 2^{24}-1 (Bit Times)
LAS	preferably maximum 32 FDL addresses (0 to 126), optionally up to 127 FDL addresses
GAPL	max. 126 FDL addresses (0 to 126) inclusive FDL Status
Counters (optional)	
Variable	**Range**
Frame_sent_count	0 to 2^{32}-1
Retry_count	0 to 2^{16}-1
SD_count	0 to 2^{32}-1
SD_error_count	0 to 2^{16}-1

5.2.3.3 Fault FDL

The service "Fault FDL" is **mandatory**.

The FDL controller delivers a FDL_FAULT.indication primitive to the FMA1/2 to inform FMA1/2 that it has detected a faulty condition or an event.

Parameters of the Primitives

FDL_FAULT.indication
(Fault_type)

- The parameter Fault_type gives the following error reasons:

Table 32. FDL Error Reasons

Name	Meaning
Duplicate_address *)	A duplicate of this station's FDL address exists.
Faulty_transceiver *)	The Transmitter or Receiver of this station is malfunctioning
Time_out	No bus activity
Not_syn	Synchronization not detected within Syn-Interval-Time T_{SYNI}
Out_of_ring *)	This Master Station was taken from the logical ring not on its own initiative
GAP_event *)	A change in the GAPL has occurred

*) only possible in Master Stations

5.3 PHY - FMA1/2 Interface

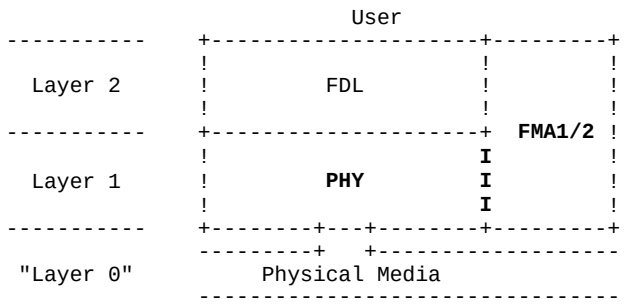


Figure 13. Interface between PHY and FMA1/2 in Relation to Layer Model

5.3.1 Overview of Services

The PHY Layer offers the following services to FMA1/2:

- Reset PHY
- Set Value PHY
- Read Value PHY
- Event PHY

Reset PHY (mandatory)

This service allows the FMA1/2 to reset the PHY Layer directly. The management receives a confirmation.

Set Value PHY (optional)

With this service the FMA1/2 is able to set certain variables in the PHY. PHY gives a confirmation to the FMA1/2 as to whether the variable could take the desired value.

Read Value PHY (optional)

With this service the FMA1/2 is able to read certain variables of the PHY. In the response the current value of the selected variable is transferred to the FMA1/2.

Event PHY (optional)

The PHY uses this service to inform the FMA1/2 about the changes in value of certain variables.

5.3.2 Overview of Interactions

The services mentioned in subclause 5.3.1 are described by following primitives (PHY_...):

Service	Primitive	permissible for following users
Reset PHY	PHY_RESET.request	Master and Slave
	PHY_RESET.confirm	- " -
Set Value PHY	PHY_SET_VALUE.request	Master and Slave
	PHY_SET_VALUE.confirm	- " -
Read Value PHY	PHY_READ_VALUE.request	Master and Slave
	PHY_READ_VALUE.confirm	- " -
Event PHY	PHY_EVENT.indication	Master and Slave

Temporal Relationships of Service Primitives:

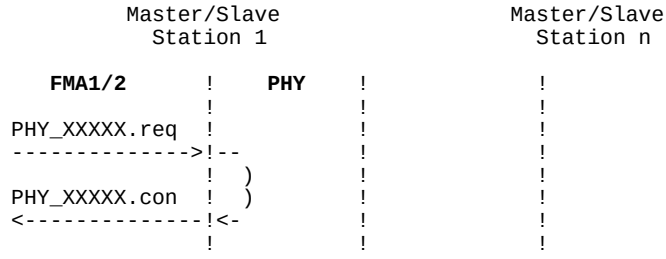


Figure 14. Reset PHY, Set Value PHY, Read Value PHY Service

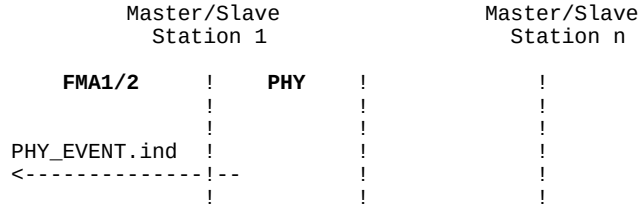


Figure 15. Event PHY Service

5.3.3 Detailed Specification of Services and Interactions

5.3.3.1 Reset PHY

The service "Reset PHY" is **mandatory**.

The PHY_RESET.request primitive is given to the PHY by the FMA1/2 to reset PHY. The PHY executes this in exactly the same manner as at a Power On (Transmitter_output: enable; Receiver_signal_source: primary; Loop: disable). An immediately returned PHY_RESET.confirm primitive informs the FMA1/2 of the executed reset.

Parameters of the Primitives:

PHY_RESET.request

PHY_RESET.confirm

- Both primitives have no parameters.

5.3.3.2 Set Value PHY, Read Value PHY

Set Value PHY

The service "Set Value PHY" is **optional**.

The FMA1/2 delivers a PHY_SET_VALUE.request primitive to the PHY to set a specified variable to a desired value. After receiving the primitive the PHY selects the variable and sets it to the value. The FMA1/2 receives a confirmation in a PHY_SET_VALUE.confirm primitive.

Parameters of the Primitives

PHY_SET_VALUE.request
(Variable_name, Desired_value)

- The parameter Variable_name specifies the following variables:

Table 33. PHY Variables (Set Value)

Name	Meaning
Transmitter_output	Transmitter Output
Received_signal_source	Receiver Input
Loop	The Transmitter Output is directed to the Receiver Input and not to the Medium

- The parameter Desired_value specifies the new value for the variable:

Table 34. Values of PHY Variables (Set Value)

Variable	Values
Transmitter_output	enabled (indirect); disabled
Received_signal_source	primary (standard source = Bus a)
	alternate (alternative source = Bus b),(see Part 2, clause 4.3)
	random (either Bus a or Bus b)
Loop	enabled; disabled

PHY_SET_VALUE.confirm

- This primitive has no parameters.

Read Value PHY

The service "Read Value PHY" is **optional**.

The FMA1/2 delivers a PHY_READ_VALUE.request primitive to the PHY to read a specified variable. The PHY transfers the value of the variable in a PHY_READ_VALUE.confirm primitive to the FMA1/2.

Parameters of the Primitives

PHY_READ_VALUE.request
(Variable_name)

- The parameter Variable_name specifies the following variables:

Table 35. PHY Variables (Read Value)

Name	Meaning
Transmitter_output	Transmitter Output
Received_signal_source	Receiver Input
Loop	internal Send Receive Loop

PHY_READ_VALUE.confirm
(Current_value)

- The parameter Current_value contains the current value which was requested with the preceding PHY_READ_VALUE.request primitive. The following values are permitted for the variables:

Table 36. Value of PHY Variables (Read Value)

Variable	Values
Transmitter_output	enabled; disabled
Received_signal_source	primary; alternate
Loop	enabled; disabled

5.3.3.3 Event PHY

The service "Event PHY" is **optional**.

The PHY uses this service to inform the FMA1/2 that a variable value was changed.

Parameters of the Primitives

PHY_EVENT.indication
(Variable_name, New_value)

- The parameter Variable_name specifies the variable whose value was changed:

Table 37. PHY Variables (Event)

Name	Meaning
Transmitter_output	Transmitter Output
Received_signal_source	Receiver Input

- The parameter New_value specifies the new value of the variable.

Table 38. Values of PHY Variables (Event)

Variable	Values
Transmitter_output	disabled; enabled
Received_signal_source	primary; alternate

PROFIBUS Specification - Normative Parts

Part 4

Data Link Layer Protocol Specification

Contents

1 Scope

(see Part 2)

2 Normative References

(see Part 2)

3 General

(see Part 2)

4 Medium Access Methods and Transmission Protocol (Data Link Layer, FDL)

A PROFIBUS System uses controlled medium access accomplished by a hybrid medium access method: a decentral method according to the principle of **Token Passing** is underlain by a central method according to the **Master - Slave** principle. Medium access control may be exercised by each master station (active station). Slave stations (passive stations) act neutrally in respect to medium access, i.e. they do not transmit independently but only on request.

Communication is always initiated by the master station which has the permission for medium access, the token. The token is passed from master station to master station in a logical ring and thus determines the instant, when a master station may access the medium. Controlled token passing is managed by each station knowing its predecessor (Previous Station, PS), the station from which it receives the token. Furthermore each station knows its successor (Next Station, NS), i.e. the station to which the token is transmitted, and its own address (This Station, TS). Each master station determines the PS and NS addresses after the initialization of the operating parameters for the first time and then later dynamically according to the algorithm described in subclause 4.1.1.2.

If the logical ring consists of only one master and several slave stations then it is a pure master - slave system.

The following error conditions, exceptions and operational states in the system are dealt with:

1) multiple tokens
2) lost token
3) error in token passing
4) duplicate station addresses
5) stations with faulty transmitter/receiver
6) adding and removing stations during operation
7) any combinations of master and slave stations

4.1 Transmission Procedures and FDL Controller

The exchange of messages takes place in cycles. A message cycle consists of a master station's action frame (request or send/request frame) and the associated acknowledgement or response frame of a master or a slave station. User data may be transmitted in the action frame (send) as well as in the response frame

(response). The acknowledgement frame does not contain any user data (for frame formats see clause 4.6).

The complete message cycle is only interrupted for token transmission and for the transmission of data without acknowledgement (e.g. necessary for broadcast messages). In both modes of operation there is no acknowledgement. In broadcast messages a master station (initiator) addresses all other stations at the same time by means of a global address (highest station address, all address bits binary "1").

All stations except the respective token holder (initiator) shall in general monitor all requests. The stations acknowledge or respond only when they are addressed. The acknowledgement or response shall arrive within a predefined time, the Slot Time, otherwise the initiator repeats the request if it is not a "first request" (see subclause 4.7.3, FCB). A retry or a new request shall not be issued by the initiator before the expiration of a waiting period, the Idle Time (see subclause 4.1.7).

If the responder does not acknowledge or respond after a predefined number of retries (see subclause 4.1.6), it is marked as "non-operational". If a responder is "non-operational", a later unsuccessful request will not be repeated.

The modes of transmission operation define the time sequence of the message cycles. Four types are distinguished:

1) Token handling

2) Acyclic request or send/request operation

3) Cyclic send/request operation, polling

4) Registration of stations

4.1.1 Token Procedures

4.1.1.1 Token Passing

The token is passed from master station to master station in ascending numerical order of station addresses by means of the token frame (see subclause 4.6.4). The one exception is that, to close the logical token ring, the station with the highest address passes the token to the station with the lowest address (see Fig.1).

Token Reception:

If a master station (TS) receives a token frame addressed to itself from a station which is registered as Previous Station (PS) in its List of Active Stations (LAS), it **owns the token** and may execute message cycles. The LAS is generated in the master station in the "Listen_Token" state (see subclause 4.1.5) after power on and updated and corrected, if necessary, later on upon receipt of a token frame.

If the token transmitter is not the registered PS, the addressee shall assume an error and not accept the token. Only a subsequent retry of the same PS is accepted and results in the token receipt, because the token receiver shall assume now that the logical ring has changed. It replaces the originally recorded PS in its LAS by the new one.

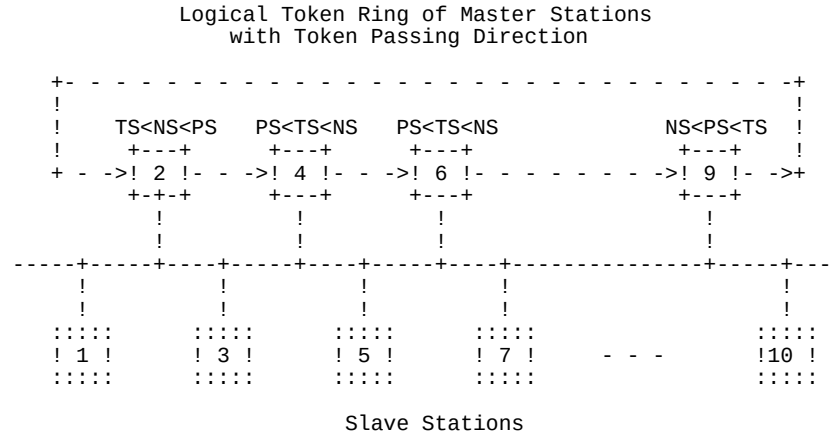


Figure 1. Logical Token Passing Ring

Token Transmission:

After the master station has finished its message cycles - contingent maintenance of the GAP Station List (GAPL, see subclause 4.1.1.2) included - it passes the token to its successor (NS) by transmitting the token frame. The functionality of its transceiver is checked by simultaneous monitoring (see subclause 4.1.5, "Pass_Token" state).

If, after transmitting the token frame and after expiration of the Syn Time within the Slot Time (see subclause 4.1.7), the token transmitter receives a valid frame, i. e. a plausible frame header without any errors, it assumes that its NS owns the token and executes message cycles. If the token transmitter receives an invalid frame, it assumes that another master station is transmitting. In both cases it ceases monitoring the token passing and retires, i.e. it enters the "Active_Idle" state (see subclause 4.1.5).

If the token transmitter does not recognize any bus activity within the Slot Time, it repeats the token frame and waits another Slot Time. It retires thereafter, if it recognizes bus activity within the second Slot Time. Otherwise it repeats the token frame to its NS for a last time. If, after this second retry, it recognizes bus activity within the Slot Time, it retires.

If, after the second retry, there is no bus activity, the token transmitter tries to pass the token to the next but one master station. It continues repeating this procedure until it has found a successor from its LAS. If it does not succeed, the token transmitter assumes that it is the only one left in the logical token ring and keeps the token or transmits it to itself, if no message cycles are requested. If it finds a NS again in a later station registration, it tries again to pass the token.

4.1.1.2 Addition and Removal of Stations

Master and slave stations may be connected to or disconnected from the transmission medium at any moment. Each master station in the logical token ring is responsible for the addition of new stations and the removal of existing stations,

the addresses of which are situated in the range from the own station address (TS) to the next station (NS). This address range is called GAP and is represented in the GAP List (GAPL), except the address range between Highest Station Address (HSA, see Part 3, Set/Read Value FDL, variables) and 127, which does not belong to the GAP.

Each master station in the logical token ring examines its address range (all GAP addresses) periodically in the interval given by the GAP Update Time (T_{GUD}) for changes concerning master and slave stations. This is accomplished by examining **one** address per token receipt, using the "Request FDL Status" action frame (see table 3a, b7=1, Code-No 9: Format 4.6.1A).

Upon receiving the token, GAP maintenance starts immediately after all queued message cycles have been conducted, if there is still transmission time available (see subclause 4.1.1.4). Otherwise GAP maintenance starts upon the next or the consecutive token receipts immediately after the high priority message cycles have been performed (see subclause 4.1.1.5). In realizations care shall be taken that GAP maintenance and low priority message cycles do not block each other.

GAP addresses are examined in ascending order, except the GAP which surpasses the HSA, i.e. the HSA and the address 0 are not used by a master station. In this case the procedure is continued at address 0 after checking the HSA. If a station acknowledges positively with the state "not ready" or "slave station" (see table 3a, b7=0, Code-No 0, no SC, and Fig. 18), it is accordingly marked in the GAPL and the next address is checked. If a station answers with the state "ready to enter logical token ring", the token holder changes its GAP or GAPL and passes the token to the new NS. This station which has newly been admitted to the logical token ring has already built up its LAS (List of Active Stations), when it was in the "Listen_Token" state, so that it is able to determine its GAP range or GAPL and its NS.

If a station answers with the state "master station in logical token ring", then for the time being the token holder does not change its GAP and passes the token to the NS given in the LAS. Thus the "jumped over" master station shall retire from the bus itself and shall enter in the "Listen_Token" state because of no correct status report. In this state it generates a new LAS and remains in this state until it is addressed once more by a "Request FDL Status" transmitted by its predecessor (PS).

Stations which were registered in the GAPL and which do not respond to a repeated "Request FDL State" are removed from the GAPL and are recorded as unused station addresses. Requests to station addresses, which have not been used so far, are not repeated.

4.1.1.3 (Re)Initializing the Logical Token Ring

Initialization is primarily a special case of updating the LAS and the GAPL. If after power on (PON) of a master station in the "Listen_Token" state a time-out is encountered, i.e. no bus activity within T_{TO} (see subclause 4.1.7), it shall claim the token ("Claim_Token" state), "take it" and start initializing.

When the entire PROFIBUS System is initialized, the master station with the lowest station address starts initialization. By transmitting two token frames addressed to itself (DA = SA = TS) it informs any other master stations (entering a NS into the LAS) that it is now the only station in the logical token ring.

Then it transmits a "Request FDL Status" frame to each station in an incrementing address sequence, in order to register other stations. If a station responds with "master station not ready" or with "slave station" it is entered in the GAPL. The first master station to answer with "ready to enter logical token ring" is registered as NS in the LAS and thus closes the GAP range of the token holder. Then the token holder passes the token to its NS.

Re-initialization becomes necessary after loss of the token. In this case an entire bus initialization sequence is not required, because LAS and GAPL already exist in the master stations. The time out expires first in the master station with the lowest address. It takes the token and starts executing regular message cycles or passes the token to its NS.

4.1.1.4 Token Rotation Time

After a master station has received the token, the measurement of the token rotation time begins. The time measurement of the expired cycle ends at the next token receipt and results in the real token rotation time T_{RR} (Real Rotation Time). At the same instant a new measurement of the following rotation time starts. T_{RR} is of significance for carrying out low priority message cycles.

In order to keep within the system reaction time required by the field of application, the Target Rotation Time T_{TR} of the token in the logical ring shall be defined.

The system reaction time is defined as the maximum time interval (worst case) between two consecutive high priority message cycles of a master station, measured at the FDL interface (see Part 3, clause 4) at maximum bus load.

Independently of the Real Rotation Time, each master station may always execute **one** high priority message cycle per token receipt.

In order to perform low priority message cycles, T_{RR} shall be less than T_{TR} at the instant of execution, otherwise the station shall retain low priority message cycles and transmit them at the next or the following token receipts.

A system's minimum Target Rotation Time depends on the number of master stations and thus on the token cycles (T_{TC}) and the duration of high priority message cycles (high T_{MC}). The predefined Target Rotation Time T_{TR} shall also contain sufficient time for low priority message cycles and a safety margin for potential retries.

In order to achieve a Target Rotation Time as short as possible, it is recommended in connection with the Application Layer 7 (APP), to declare only rarely occurring and important events (see Part 3, subclause 4.1.3.1) as high priority message cycles and to strictly restrict their length (e.g. ≤ 20 octets for the DATA_UNIT, see clause 4.5).

If the cycle times defined in clause 4.2 (formulas (21) and (22)) are included and possible retries are taken into consideration, the operating parameter "Target Rotation Time T_{TR}" (see subclause 4.1.7), which is necessary for initialization, is calculated as follows:

min T_{TR} =

$$na \cdot (T_{TC} + \text{high } T_{MC}) + k \cdot \text{low } T_{MC} + mt \cdot \text{RET } T_{MC} \quad (1)$$

Explanations:

na	number of master stations
k	estimated number of low priority message cycles per token rotation
T_{TC}	token cycle time
T_{MC}	message cycle time, depending on frame length (see subclause 4.2.2)
mt	number of message retry cycles per token rotation
RET T_{MC}	message retry cycle time

The first term contains **one** high priority message cycle per master station and token rotation. Thus the maximum reaction time for high priority message cycles without retry cycles is guaranteed for all bus loads. The second term contains the estimated number of low priority message cycles per token rotation. The third term serves as a safety margin for potential retries.

4.1.1.5 Message Priorities

In the service classes for message cycles the user of the FDL interface (application layer) may choose two priorities: "low" and "high". The priority is passed to the FDL with the service request.

When a master station receives the token, it always performs all available high priority message cycles first and then the low priority ones. If at the receipt of the token the Real Rotation Time T_{RR} is equal to or greater than the Target Rotation Time T_{TR}, only **one** high priority message cycle including retries in the case of an error may be performed. Then the token shall be passed to NS immediately.

In general after the receipt of the token or after the first high priority message cycle the following is to be considered: high priority or low priority message cycles may be carried out only if at the beginning of the execution T_{RR} is less than T_{TR} and thus the Token Holding Time $T_{TH} = T_{TR} - T_{RR}$ is still available. Once a high or low priority message cycle is started it is always completed, including any required retry (retries), even if T_{RR} reaches or exceeds the value of T_{TR} during the execution. The prolongation of the Token Holding Time T_{TH} caused thereby automatically results in a shortening of transmission time for message cycles at the next token receipt.

4.1.2 Acyclic Request or Send/Request Mode

In the Acyclic Request or Send/Request Mode single message cycles are conducted sporadically. The master station's FDL Controller initiates this mode due to a local user's request upon receipt of the token. If there are several requests, this mode of operation may be continued until the maximum allowed token rotation time expires.

4.1.3 Cyclic Send/Request Mode

When polling, the master station cyclically addresses stations with the request "Send and Request Data low" (see table 3a), according to a predefined sequence, the Poll List. The Poll List is passed to the FDL Controller by the local FDL user. All slave and master stations to be polled are marked in this list. The

stations which do not answer during the polling in spite of retries are marked as "non-operational". In later request cycles these stations are requested on trial, without any retry. If stations answer in this procedure, they are marked as "operational".

After receipt of the token, handling of the Poll List (Poll Cycle) is only started after all requested high priority message cycles have been carried out.

If required, polling is underlain with some additional low priority message cycles such as the Acyclic Request or Send/Request Mode, station registration (Live List) and GAP maintenance.

After each complete Poll Cycle the requested low priority message cycles are performed in turn. The order in which the message cycles are performed obeys the following rules:

If the Poll Cycle is completed within the Token Holding Time T_{TH}, i.e. there is still Token Holding Time available, the requested low priority message cycles are carried out in turn as far as possible within the remaining Token Holding Time. A new Poll Cycle starts at the next receipt of the token which has Token Holding Time for low priority message cycles available.

If at the end of a Poll Cycle there is not any further Token Holding Time available, the requested low priority message cycles are as far as possible processed at the next token receipt that has available Token Holding Time for low priority message cycles. After that, a new Poll Cycle starts as described above.

If a Poll Cycle takes several Token Holding Times, the Poll List is processed in segments, but without inserting requested low priority message cycles. Low priority message cycles are carried out only at the end of a complete Poll Cycle, as described above.

The low priority message cycles which underlie the polling are performed in the order of their arrival. Thereby for GAP maintenance at most **one** address of the GAPL shall be checked between Poll Cycles as described in subclause 4.1.1.2.

The Poll Cycle time, i.e. the maximum station delay time, depends on the duration of a message cycle (see clause 4.2), on the token rotation time, on the length of the Poll List and on the underlain low priority message cycles. With multiple entries of a few individual stations in the Poll List the request priority of these stations may be increased, and thus their reaction time shortened.

4.1.4 Request FDL Status of all Stations (Live List)

The FDL Controller enters this mode of operation if the local user requests a Live List via management (FMA1/2). At token receipt the mode starts after performing any previously requested low priority message cycles. During polling the mode is performed between Poll Cycles. A cyclic "Request FDL Status" is used (see table 3a, b7=1, Code-No 9). According to the given FDL address space (DA = 0 to 126, see subclause 4.7.2) each possible station is addressed once, except the master stations registered in the LAS. The correctly responding stations, i.e. those which acknowledge positively (see table 3a, b7=0, Code No 0, no SC), and the master stations of the LAS are entered in the Live List as existing master or slave stations (see subclause 4.7.3, Station Type). The Live List is formally structured as follows:

Table 1. Live-List

Entry	Name
1	Length of Live-List = 3 to 2n+1
2	FDL Address (DA) of Station k
3	Station Type and FDL Status k
4	DA of Station k+1
5	Station Type and FDL Status k+1
.	...
.	...
.	...
l	DA of Station n
l+1	Station Type and FDL Status n
k: first live Station; n ≤ 127 ; l ≤ 254	

4.1.5 Status of the FDL Controller

A master station's FDL Controller (in the following denoted FDL) is described by means of 10 FDL states and the transitions between them. A slave station has two FDL states.

Fig. 2 shows as an overview the combined FDL state diagram of the master (state 0 to 9) and the slave station (state 0 and 10).

Offline

The "Offline" state shall be entered immediately after power on, after the FMA1/2 service "Reset FDL" (see Part 3, subclause 5.2.3.1), or after certain error conditions have been detected. After power on each station should perform a self-test. This internal self-test depends on the implementation and does not influence the other stations. For this reason, the self-test procedure is not specified in this specification.

After terminating the power on sequence, the FDL remains in the "Offline" state until all required operating parameters (see subclause 4.1.6) have been initialized. The FDL may only then connect to the transmission medium, but without transmitting itself.

Passive_Idle

After its parameters are initialized, the slave station's FDL shall enter the "Passive_Idle" state and listen to the line. If a plausible action frame (send/request frame), addressed to that station, is received, the FDL shall acknowledge or respond as required, except for frames with global address (broadcast message, see subclause 4.7.2) and token frames addressed to itself. The token frame is discarded.

At the occurrence of the FMA1/2 service "Reset FDL", and if a fatal error (e.g. continuous transmission) is detected, the FDL re-enters the "Offline" state.

Listen_Token

After its operating parameters have been initialized, the master station's FDL shall enter the "Listen_Token" state, if it is ready to enter the logical token ring. In this state the master station's FDL shall monitor the line in order to identify those master stations which are already in the logical token ring. For that purpose token frames are analyzed and the station addresses contained in

them are used to generate the list of active stations (LAS). After listening to two complete identical token rotations, the FDL shall remain in the "Listen_ Token" state until it is addressed by a "Request FDL Status" transmitted by its predecessor (PS). It shall respond with "ready to enter logical token ring" and at receiving the next token frame addressed to itself, it shall enter the "Active_Idle" state. During LAS generation any "Request FDL Status" is not acknowledged or responsed with "not ready". All other frames are not processed in the "Listen_Token" state, i.e. they are neither acknowledged nor answered.

If the FDL detects its own address as source address (SA) in two token frames when registering the master stations, it shall assume that another master station with the same address exists already in the ring. The FDL shall then re-enter the "Offline" state and report this to management (FMA1/2).

If the FDL observes no bus activity for the time-out period, it shall assume that an initialization or a restoration of the logical token ring is necessary. The FDL tries to claim the token and to (re)initialize the logical ring.

Active_Idle

On leaving the "Listen_Token" state, the master station's FDL shall enter the "Active_Idle" state and listen to the bus line without becoming active. If it receives a plausible action frame addressed to itself, it shall acknowledge or respond as required. After receiving a token frame addressed to itself, it shall enter the "Use_Token" state, if the station wants to remain in the logical token ring, otherwise it shall re-enter the "Listen_Token" state. This state shall also be entered in the case of an error, if two token frames with SA = TS are received in immediate succession.
If the FDL find out that it was taken from the logical token ring not on its own initiative, it also shall enter in the "Listen_Token" state and report this (Out_of_ring) to management (FMA1/2).

If the FDL observes no bus activity for the time-out period, it shall assume that a restoration of the logical token ring is necessary. The FDL tries to claim the token and to (re)initialize the logical ring ("Claim_Token" state).

Claim_Token

The FDL shall enter the "Claim_Token" state after the "Active_Idle" state and the "Listen_Token" state, when its time-out has expired. In this state it shall try to re-initialize the logical ring or to start an initialization. When re-initializing, the stations' status lists (LAS and GAPL) are still available, and thus the "Use_Token" state is entered immediately.

When initializing, at first the token shall be addressed twice to the own FDL, i.e. NS = TS, namely in the "Pass_Token" state. This is necessary in order to cause an entry in the other master stations' LAS. After token transmission the own GAPL and the NS shall be created in the "Await_Status_Response" state by means of "Request FDL Status" requests to the following station addresses.

Use_Token

The FDL shall enter the "Use_Token" state after receiving a token or after (re)initialization. This is the state in which the FDL may carry out high priority and low priority message cycles.

On entering this state T_{RR} (Real Rotation Time) shall be read from the Token Rotation Timer and the timer shall be restarted. **One** high priority message cycle is always possible. A further high priority or low priority message cycle or generally a low priority message cycle may only be carried out if $T_{RR} < T_{TR}$ (Target Rotation Time) at the instant of execution.

After each transmitted action frame the FDL shall enter the "Await_Data_Response" state and start the Slot Timer (see subclause 4.4). It may return to the "Use_Token" state as soon as the frame transmitted before has been confirmed to the user.

The FDL shall enter the "Check_Access_Time" state, if at the beginning of "Use_Token" no high priority message cycle is to be performed, or after the completion of a high priority or low priority message cycle.

Await_Data_Response

This state shall be entered after the transmission of an action frame. The FDL waits **one** Slot Time for the receipt of the acknowledgement or response frame.

In the case of a SDN service (Send Data with No Acknowledge) no acknowledgement is awaited. The FDL re-enters the "Use_Token" state in order to process potential further requests. In the case of a request with acknowledgement or response the FDL shall wait for one of the following events:

a) a valid acknowledgement frame or valid response frame addressed to the request's initiator.

b) any other valid frame (e.g. token frame or action frame)

c) invalid frame (start-, end-, length-, FCS-byte error; start-, stop-, parity-bit error or slip error) or Slot Time expired (see subclause 4.1.7).

After receiving and processing an acknowledgement or response frame, the FDL shall re-enter the "Use_Token" state in order to process potential further service requests.

Receipt of another valid frame indicates that an error has occurred. The FDL shall enter the "Active_Idle" state and discard the received frame.

If an invalid frame is received or the Slot Time expires, the FDL shall retry the transmission of the action frame. If after the retry (retries) no valid acknowledgement or response is received, the FDL shall notify the user accordingly and re-enter the "Use_Token" state. Further requests to this station are not repeated in case of errors, until a correct message cycle (Send/Request Data) has been performed.

Check_Access_Time

In this state the available Token Holding Time shall be computed by means of the difference $T_{TR} - T_{RR}$. Only if there is still Token Holding Time available, the FDL may re-enter the "Use_Token" state. Otherwise the FDL shall enter the "Pass_Token" state.

Pass_Token

In the "Pass_Token" state the FDL shall try to pass the token to the next station (NS) in the logical ring. When transmitting the token frame, the FDL shall

check by simultaneous monitoring if the transceiver is working correctly. If it does not receive its own token frame, there is a fatal error in the transmit or receive channel. The FDL shall stop its activity in the logical ring, enter the "Offline" state and notify management (FMA1/2).

If the FDL receives its own token frame corrupted, this may be caused by a temporarily defective transmitter, receiver, or by the bus line. This error condition does not result in stopping activities in the first instance, instead the FDL shall enter the "Check_Token_Pass" state as after receiving the token frame correctly. Only after the token frame has been retransmitted (due to no reaction from NS) and monitored as incorrect, shall the FDL stop its activity in the logical ring, enter the "Listen_Token" state and notify the management (FMA1/2).

When the GAP Update Time has expired, but there is still Token Holding Time T_{TH} available, before passing the token the FDL shall try to record one possible new station in the GAP, in order to include it in the logical ring, if necessary. For that purpose FDL transmits a "Request FDL Status" and enters the "Await_Status_Response" state. If then a new master station acknowledges that it wants to be included in the logical ring, the FDL shall pass the token to this station. After the token has been successfully passed, the own GAP is shortened to the new station.

If a status request is answered by a new slave station or a master station that does not want to be included in the ring, that station shall be entered in the GAPL. If an existing station does not answer even after a retry, it shall be deleted from the GAPL, i.e. be marked as an unused address.

If during GAP maintenance no new master station answers, the FDL shall pass the token to its original NS and enter the "Check_Token_Pass" state. Only if no successor is known, i.e. the FDL is at the moment the only active station on the bus, it shall pass the token to itself and then re-enter the "Use_Token" state.

Check_Token_Pass

"Check_Token_Pass" is the state in which the FDL waits **one** Slot Time for a reaction of the station to which it has passed the token. This waiting time allows for the delay between the receipt of the token frame and the ensuing transmission reaction of the addressed station.

If the FDL detects a valid frame header within one Slot Time, it shall assume that the token passing was successful. The frame shall be processed as if it were received in the "Active_Idle" state, i.e. the FDL shall enter this state.

If an invalid frame is detected within the Slot Time, the FDL shall assume that another station is active and therefore also enter the "Active_Idle" state.

If the FDL does not receive any frame within one Slot Time, it shall re-enter the "Pass_Token" state and react as described in subclause 4.1.1.1.

Await_Status_Response

This state is entered from the "Pass_Token" state after it has been decided that a successor is not known, as e.g. during initialization or GAP maintenance. In this state the FDL shall wait **one** Slot Time for an acknowledgement frame. If nothing or a corrupted frame is received, the FDL shall re-enter the "Pass_Token" state, in order to repeat either the request or to pass the token to itself or to its successor.

If the FDL receives any other frame instead of an acknowledgement frame (indicates that multiple tokens may exist), it shall enter the "Active_Idle" state.

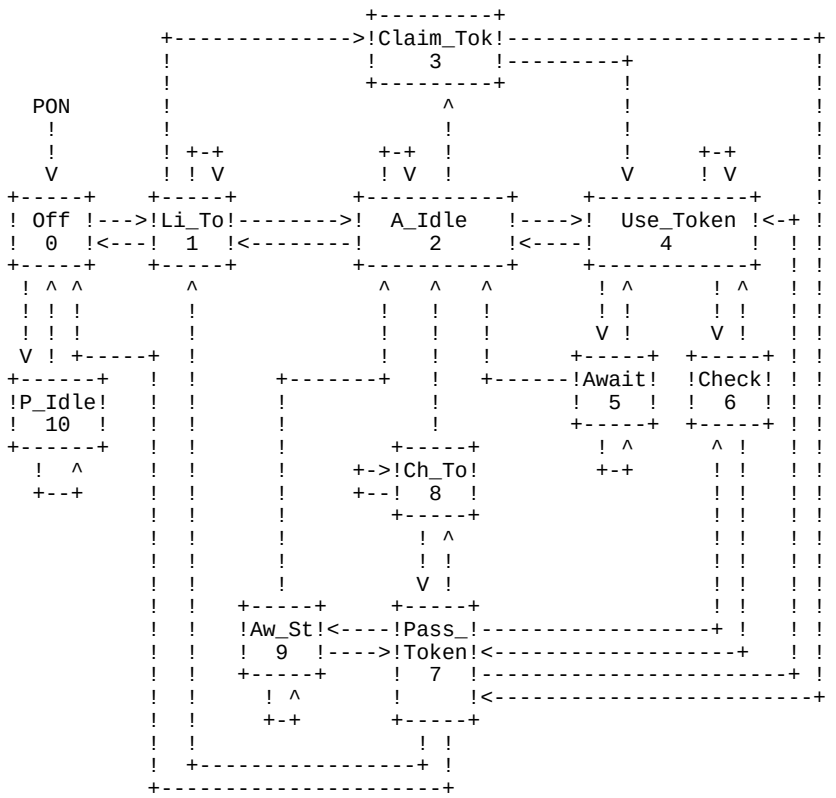


Figure 2. FDL State Diagram

4.1.6 FDL Initialization

After power on (PON) the FDL Controller of master and slave stations shall enter the "Offline" state. Within this state the FDL Controller does not receive or transmit any signals (frames) from or to the bus line respectively.

The FDL Controller may enter the "Passive_Idle" or "Listen_Token" state from the "Offline" state only, if the operating parameters (FDL Variables) have been set for correct protocol handling (see Part 3, subclause 5.2.3, Set Value FDL). The following operating parameters shall be provided by management (FMA1/2):

Table 2. Operating Parameters

Ser. No	Name
1	Station Address TS
2	Data Signalling Rate (Baud_rate; kbit/s)
3	Single/Redundant Media available
4	Release of Hardware and Software
5	Slot Time T_{SL}
6	Station Delay Time min T_{SDR}
7 *)	Station Delay Time max T_{SDR}
8 *)	Transmitter fall/Repeater switch Time T_{QUI}
9 *)	Setup Time T_{SET}
10 *)	Target Rotation Time T_{TR}
11 *)	GAP Update Factor G
12 *)	Master Station enter/leave the Logical Ring
13 *)	Highest Station Address (HSA)
14 *)	Maximum Number of Retries (max_retry_limit)

*) applies only to Master Stations

4.1.7 Timer Operation

The following times T are measured in bits. A time t in seconds (s) shall therefore be divided by the bit time t_{BIT}.

Bit Time t_{BIT}:

The Bit Time t_{BIT} is the time which elapses during the transmission of one bit. It is equivalent to the reciprocal value of the transmission rate:

$$t_{BIT} = 1 / \text{Transmission Rate in bit/s.} \quad (2)$$

Syn Time T_{SYN}:

The Synchronization Time T_{SYN} is the minimum time interval during which each station shall receive idle state (idle = binary "1") from the transmission medium before it may accept the beginning of an action frame (request or send/request frame) or token frame. The Syn Time equals:

$$T_{SYN} = 33 \text{ bit} \quad (3)$$

Syn Interval Time T_{SYNI}:

The Synchronization Interval Time T_{SYNI} serves the supervision of the maximum allowed time interval between two consecutive Syn Times or receiver synchronizations. This time comprises two complete message cycles, each of which consists of two frames of maximum length and the related Syn Times. A transmission disturbance is permitted in one of those Syn Times:

T_{SYNI} =

$$2 \cdot (2 \cdot (33 \text{ bit} + 255 \cdot 11 \text{ bit})) + 33 \text{ bit} = 11\,385 \text{ bit} \quad (4)$$

Station Delay Time T_{SDX}:

The Station Delay Time T_{SDX} is the period of time which may elapse between the transmission or receipt of a frame's last bit until the transmission or receipt of a following frame's first bit (with respect to the transmission medium, i.e. including line receiver and transmitter). The following three station delays are defined:

1) Station Delay of Initiator (station transmitting action or token frame):

$$\mathbf{T_{SDI}} = t_{SDI} / t_{BIT} \quad (5)$$

2) Minimum Station Delay of Responders (stations which acknowledge or respond):

$$\mathbf{min\ T_{SDR}} = \min t_{SDR} / t_{BIT} \quad (6)$$

3) Maximum Station Delay of Responders:

$$\mathbf{max\ T_{SDR}} = \max t_{SDR} / t_{BIT} \quad (7)$$

When transposing the NRZ signals into a different signal coding, the transmitter fall time after switching off the transmitter (at the initiator) shall be taken into account if it is greater than T_{SDR}. During this Quiet Time T_{QUI} transmission and receipt of frames shall be disabled. This shall also be taken into account when using self-controlled repeaters, whose switching time shall be taken into consideration. For T_{QUI}:

$$\mathbf{T_{QUI}} < \min T_{SDR} \quad (8)$$

In order to fulfil condition (8), it may be necessary to prolong T_{SDR}.

Ready Time T_{RDY}:

The Ready Time T_{RDY} is the time within which a master station shall be ready to receive an acknowledgement or response after transmitting a request. The Ready Time is defined as follows:

$$\mathbf{T_{RDY}} < \min T_{SDR} \quad (9)$$

In order to fulfil this condition it may be necessary to prolong T_{SDR}.

When transposing NRZ signals into a different signal coding, the Quiet Time shall also be taken into consideration when switching off the transmitter. The receiver shall not be enabled before this time:

$$\mathbf{T_{QUI}} < T_{RDY} \quad (10)$$

In order to fulfil this condition, it may be necessary to prolong T_{RDY} and thus T_{SDR} accordingly.

Safety Margin T_{SM}:

The following time interval is defined as Safety Margin T_{SM}:

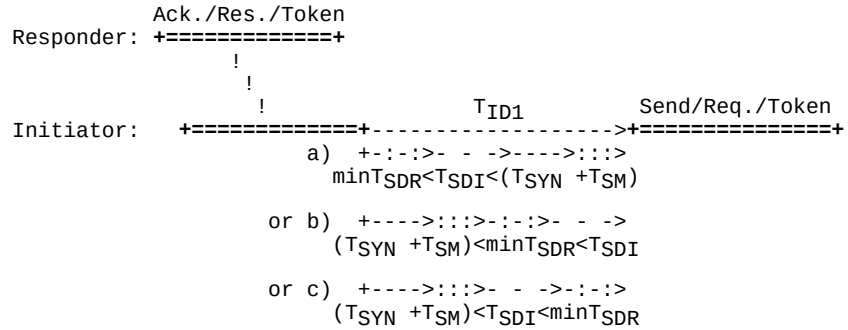
$$\mathbf{T_{SM}} = 2\ \text{bit} + 2 \cdot T_{SET} + T_{QUI} \quad (11)$$

T_{SET} is the set-up time which expires from the occurrence of an event (e.g. interrupt: last octet sent or Syn Time expired) until the necessary reaction is performed (e.g. to start Syn Time or to enable the receiver):

$$T_{SET} = t_{SET} / t_{BIT} \quad (12)$$

Idle Time T_{ID}:

The Idle Time T_{ID} is the time which expires at the initiator after receipt of a frame's last bit (measured at the line receiver) as idle = binary "1" **on the transmission medium**, until a new frame's first bit is transmitted on the medium (including line transmitter). The Idle Time is also the time which expires between transmitting the last bit of a frame which is not to be acknowledged and transmitting the first bit of the next frame. The Idle Time shall be least the Syn Time plus the Safety Margin T_{SM} (see Fig. 3 and Fig. 4, case a)). At high transmission rates (see Fig. 3 and Fig. 4, case b)and c)) the Syn Time is very short, hence the station delays become significant and shall be taken into consideration. Two Idle Times are distinguished. After an acknowledgement, response or token frame the Idle Time is defined as follows:



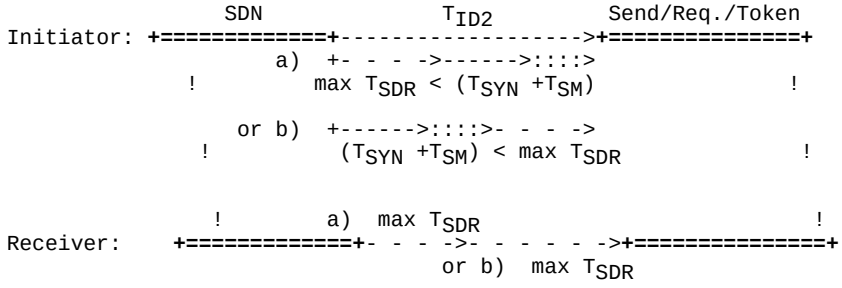
$$T_{ID1} = \max (T_{SYN} + T_{SM} , \min T_{SDR} , T_{SDI})^{*)} \quad (13)$$

*) the maximum of the three values.

Figure 3. Idle Time T_{ID1}

The parameter min T_{SDR} is extreme short to define because of the dynamic of the system, but to select greater than the Ready Time T_{RDY}. If the necessary delay time cannot be reached with the range of value of T_{SET}, than the min T_{SDR} shall be made longer.

After an action frame, which is **not** to be acknowledged (Send Data with No Acknowledge, SDN) the Idle Time is defined as:



$$\mathbf{T_{ID2}} = \max \left(T_{SYN} + T_{SM} , \max T_{SDR}\right)^{*)} \quad (14)$$

*) the maximum of the two values.

Figure 4. Idle Time T_{ID2}

Transmission Delay Time T_{TD}:

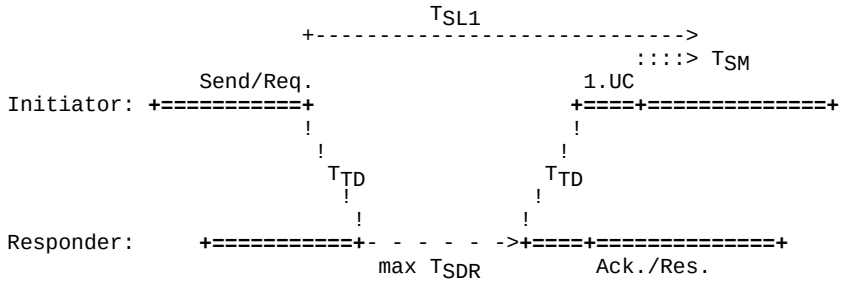
The Transmission Delay Time T_{TD} is the maximum time which elapses on the transmission medium between transmitter and receiver when a frame is transmitted. Delay times of repeaters shall be considered, if necessary. The Transmission Delay Time is defined as follows:

$$\mathbf{T_{TD}} = t_{TD} / t_{BIT} \quad (15)$$

e.g. given a line length of 200 m without repeaters, t_{TD} is approx. 1 µs and thus at 500 kbit/s T_{TD} = 0,5 bit.

Slot Time T_{SL}:

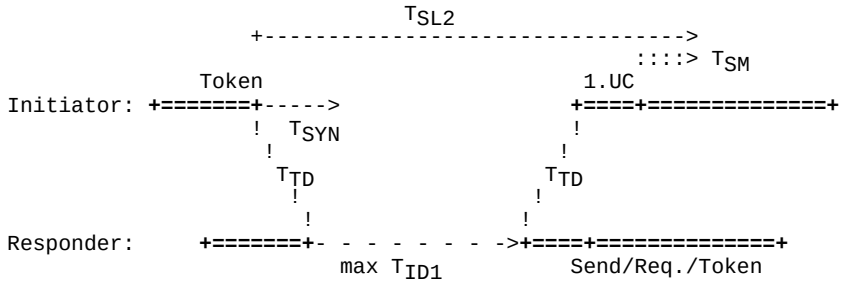
The Slot Time T_{SL} is the maximum time the initiator waits for the complete receipt of the first frame character (see subclause 4.5.1, UART Character, 1 UC = 11 bits) of the immediate acknowledgement or response, after transmitting the last bit of an action frame (including the line transmitter). Furthermore, T_{SL} is the maximum time the initiator waits for the token receiver's first frame character after transmitting a token frame. Theoretically two Slot Times are distinguished. After an action frame (request or send/request) the Slot Time is calculated as follows:



$$T_{SL1} = 2 \cdot T_{TD} + \max T_{SDR} + 11 \text{ bit} + T_{SM} \quad (16)$$

Figure 5. Slot Time T_{SL1}

After a token frame the Slot Time is calculated as follows:



$$T_{SL2} = 2 \cdot T_{TD} + \max T_{ID1} + 11 \text{ bit} + T_{SM} \quad (17)$$

Figure 6. Slot Time T_{SL2}

In order to simplify the realization, only **one** Slot Time, the longer one, is used in the system. This does not influence the system reaction time negatively, as the Slot Time is a pure monitoring time.

$$T_{SL} = \max (T_{SL1} , T_{SL2})^{*)} \quad (18)$$

*) the larger value shall be chosen

Time-out T_{TO}:

The time-out T_{TO} serves to monitor the master and slave station's bus activity and Idle Time. Monitoring starts after PON, immediately in the "Listen_Token" or "Passive_Idle" state or later after receiving the last bit of a frame. It ends after receiving the first bit of a following frame. If the Idle Time reaches time-out, the bus is regarded as inactive (error, e.g. due to lost token). The time-out is defined as follows:

$$T_{TO} = 6 \cdot T_{SL} + 2 \cdot n \cdot T_{SL} \quad (19)$$

For master stations: n = station address (0 to 126)

For slave stations: n = 130, independent of its station address

The first term makes sure that there is sufficient difference to the maximum possible Idle Time between two frames. The second term ensures that not all master stations claim the token at the same moment after an error has occurred.

GAP Update Time T_{GUD}:

The GAP Update Time T_{GUD} serves the master station for initializing GAP maintenance. After the first generation of the GAPL, updating the GAP image is cyclically initialized after every interval T_{GUD}. This initialization takes place at the next possible token receipt, if there is still Token Holding Time available after the regular message cycles, or during later token holding phases. The GAP Update Time is a multiple of the Target Rotation Time T_{TR} and is defined as follows:

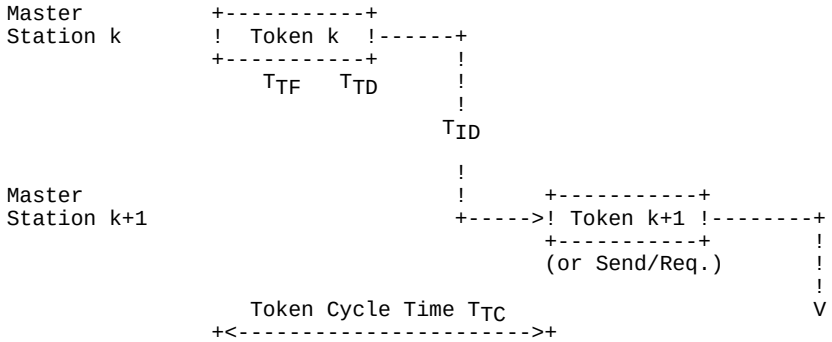
$$T_{GUD} = G \cdot T_{TR} \qquad 1 \leq G \leq 100 \quad (20)$$

T_{TR} is the predefined Target Rotation Time (see subclause 4.1.1.4).

4.2 Cycle and System Reaction Times

4.2.1 Token Cycle Time

The base load in a system with several master stations, i.e. the bus load caused by medium access control (token frames) and not by regular message cycles, is determined by the Token Cycle TC. The total base load per token rotation results from na (number of master stations) token cycles. The cycle time T_{TC} is composed of the Token Frame Time T_{TF}, the Transmission Delay Time T_{TD} and the Idle Time T_{ID}. T_{ID} results from the Station Delay Time T_{SD} or the Syn Time T_{SYN} respectively. T_{TC} is measured in bits:



$$T_{TC} = T_{TF} + T_{TD} + T_{ID} \quad (21)$$

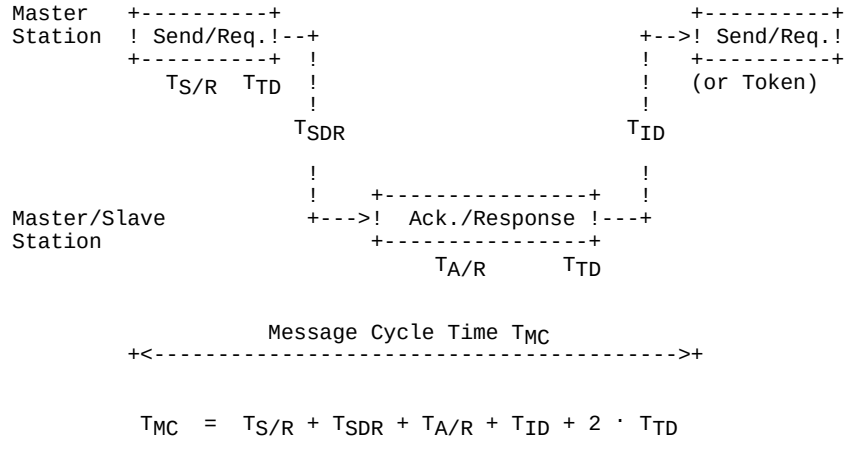
Figure 7. Token Cycle

The Token Frame Time T_{TF} is determined by the number of frame characters UC (UART character). A frame character always consists of 11 bits (see subclause 4.5.1) and hence the token frame comprises altogether 33 bits. The Transmission Delay Time T_{TD} depends on the line length (about 5 ns/m without repeater) and is mostly substantially less than the other times. The Idle Time T_{ID}, which elapses between the token frames, contains the Station Delay Time T_{SDI} of the token receiver on the one hand, on the other hand the Syn Time T_{SYN} + T_{SM} shall be used, if this sum is larger than T_{SDI} (see subclause 4.1.7). This mainly occurs at low transmission rates (< 100 kbit/s).

4.2.2 Message Cycle Time

A message cycle MC consists of the action frame (request or send/request frame) and the reply frame (acknowledgement or response frame). The cycle time is composed of the frame transmission times, the transmission delay times and the station delay times.

The Station Delay Time T_{SDR} elapses between request and acknowledgement or response. This time is needed for decoding the request and assembling the acknowledgement or response frame. It depends on the protocol implementation in the station and is mostly substantially greater than the Transmission Delay Time T_{TD}. The Idle Time T_{ID}, which elapses between acknowledgement or response and new request, contains also the Station Delay Time (see subclause 4.1.7). However, T_{SYN} + T_{SM} shall be used, if this sum is greater than T_{SDI}.



$$T_{MC} = T_{S/R} + T_{SDR} + T_{A/R} + T_{ID} + 2 \cdot T_{TD} \quad (22)$$

Figure 8. Message Cycle

At transmission rates < 100 kbit/s decoding and evaluation of frames may partially keep pace with their reception. Station delay times become considerably shorter then.

The frame transmission times ($T_{S/R}$, $T_{A/R}$) are determined by the number of frame characters (UC). Thus they are calculated as follows:

$T_{S/R}$ = a · 11 bit a No. of UC in Send/Request Frame

$T_{A/R}$ = b · 11 bit b No. of UC in Ack./Response Frame

EXAMPLE:

a = 6 for the request frame:

$T_{S/R}$ = 66 bit

b = 59 for the response frame (50 octet DATA_UNIT):

$T_{A/R}$ = 649 bit

4.2.3 System Reaction Times

The message rate R_{SYS} in the system equals the possible number of message cycles per second:

$$R_{SYS} = 1 / t_{MC} \quad ; \quad t_{MC} = T_{MC} \cdot t_{BIT} \tag{23}$$

The maximum system reaction time T_{SR} in a system with **one** master station and n slave stations (master slave system) in pure polling mode is calculated from the message cycle time and the number of slave stations. If message retries are allowed for, T_{SR} is calculated as follows:

$$T_{SR} = np \cdot T_{MC} + mp \cdot RET\ T_{MC} \tag{24}$$

where:
np number of slave stations
mp number of message retry cycles per Poll Cycle
RET T_{MC} message retry cycle time

The maximum System Reaction Time in a system with **several** master stations and slave stations equals the Target Rotation Time:

$$T_{SR} = T_{TR} \quad \text{(see subclause 4.1.1.4)} \tag{25}$$

4.3 Error Control Procedures

Line protocol errors, such as e.g. frame errors, overrun errors and parity errors, and transmission protocol errors, such as e.g. faulty start delimiters, frame check octets and end delimiters, invalid frame length, response times, etc., result in the following station reactions:

An action frame (request, send/request or token) that has been received incorrectly by a station shall not be processed, acknowledged or answered. The initiator shall retry the request after expiration of the Slot Time. The request shall also be retried if the acknowledgement or response was corrupted. The initiator shall complete a request only after having received a valid response or if the retry (retries) was not (were not) successful (see subclause 4.1.6, operating parameters). This means that a "Send/Request" shall be kept until the responder confirms

its correct receipt by an acknowledgement or response or the retry (retries) was not (were not) successful. In the same way, a responder shall terminate a "Request" or "Send/Request" only if a new request with altered Frame Count Bit is received or another station is addressed (see subclause 4.7.3, FCB).

If a station does not acknowledge or respond in cyclic or acyclic mode after retry (retries), it is marked as "non operational". When processing the following

requests, the initiator transmits the request to this station without retry (retries), until the station acknowledges or responds correctly again. After positive acknowledgement the initiator marks again the addressed station as "operational". When processing the next request, the initiator continues the original mode of operation with this station.

4.4 Timers and Counters

In order to measure the token rotation time and to realize the supervisory times the following timers are necessary:

Token Rotation Timer, Idle Timer, Slot Timer, Time-out Timer, Syn Interval Timer and GAP Update Timer.

Token Rotation Timer: When a master station receives the token, this timer is loaded with the Target Rotation Time T_{TR} and decremented each bit time. When the station again receives the token, the timer value, the remaining time or Token Holding Time T_{TH}, is read and the timer reloaded with T_{TR}. The Real Rotation Time T_{RR} results from the difference T_{TR} - T_{TH}. Low priority message cycles may be processed if at the instant of processing $T_{RR} < T_{TR}$.

Idle Timer: This timer monitors the idle state (binary "1"), the Syn Time, immediately on the bus line. The Syn Time preceding each request is necessary for unambiguous receiver synchronization. The Idle Timer of slave stations and master stations "without token" is loaded with T_{SYN} after the transmission or receipt of a frame's last bit and then decremented each bit time. The receiver shall be enabled immediately after the timer has expired. The timer of a master station "with token" is loaded according to the data transmission service with T_{ID1} or T_{ID2} (see subclause 4.1.7). A new request or token frame may only be transmitted after expiration of the timer. When the signalling level is binary "0", the timer is always reloaded.

Slot Timer: This timer in a master station monitors after a request or token pass whether the receiving station responds or becomes active within the predefined time T_{SL}, the Slot Time. After transmission of a frame's last bit this timer is loaded with T_{SL} and decremented each bit time as soon as the receiver is enabled. If the timer expires before a frame's first bit is received, an error has occurred. Then a retry or a new message cycle is initiated.

Time-out Timer: This timer monitors bus activity in master and slave stations. After the transmission or receipt of a frame's last bit the timer is loaded with a multiple of the Slot Time (see subclause 4.1.7) and decremented each bit time as long as no new frame is received. If the timer expires, a fatal error has occurred, which for the master station causes a (re)initialization. The FMA1/2 User of the slave and master station receives a time-out notification (see Part 3, subclause 4.2.3.3).

Syn Interval Timer: Master and slave stations use this timer to monitor the transmission medium as to whether a receiver synchronizing (T_{SYN}, idle state, idle = binary "1") occurs within T_{SYNI}. Each time the receiver is synchronized, the timer is loaded with T_{SYNI} (see subclause 4.1.7). From the beginning of a frame (first start bit) the timer is decremented each bit time as long as no new T_{SYN} is detected. If the timer expires, an error has occurred on the transmission medium, e.g. stuck at "0" or permanent "0" / "1" edges. The FMA1/2 User is notified accordingly (see Part 3, subclause 4.2.3.3).

GAP Update Timer: Only master stations need this timer. Its expiration indicates the moment for GAP maintenance. After a complete GAP check, which may last several token rotations (segmented; see subclause 4.1.1.2), the timer is loaded with a multiple of the Target Rotation Time T_{TR} (see subclause 4.1.7).

When a master station enters the "Listen_Token" state, the Idle Timer is loaded with T_{SYN}, the Time-out Timer with T_{TO}, the Syn Interval Timer with T_{SYNI} and the other timers are cleared. When a slave station enters the "Passive_Idle" state, the Time-out Timer is loaded with T_{TO} and the Syn Interval Timer with T_{SYNI}.

For installation and maintenance the following counters (FDL variables) are **optionally** defined in pairs:

For master stations:

- counter for transmitted frames (Frame_sent_count for Request frames) except for SDN service and FDL status
- counter for transmitted frame retries (Retry_count)

For slave and master stations:

- counter for received valid start delimiters (SD_count)
- counter for received invalid start delimiters (SD_error_count)

When a station enters the "Listen_Token" or "Passive_Idle" state, the counters are cleared and enabled. If a counter reaches its maximum (see Part 3, subclause 5.2.3.2, table "Additional FDL Variables"), counting of this counter as well as of the related comparative counter is stopped. When clearing a counter the related comparative counter is also cleared and they are enabled again. The FMA1/2 user may access these counters using the Set/Read Value FMA1/2 services (see Part 3, clause 4.2).

In relation to stations multiple counters (statistics) are possible (optional, see part 8).

4.5 Frame Structure

4.5.1 Frame Character (UART Character)

Each frame consists of a number of frame characters, the UART characters. The UART character (UC) is a start-stop character for asynchronous transmission and is structured as follows:

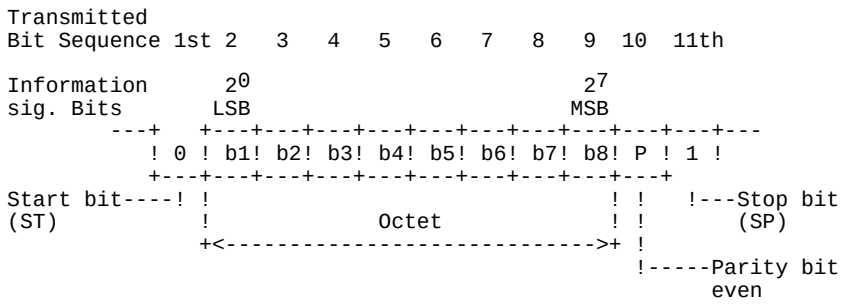


Figure 9. UART Character

The presentation of UART characters is based on the following standards: ISO 1177, ISO 2022.

Transmission Rule

Each UART character consists of 11 bits: a start bit (ST) which is always binary "0", 8 information bits (I) which may be binary "0" or binary "1", an even parity bit (P) which is binary "0" or binary "1" and a stop bit (SP) which is always binary "1".

4.5.2 Bit Synchronizing

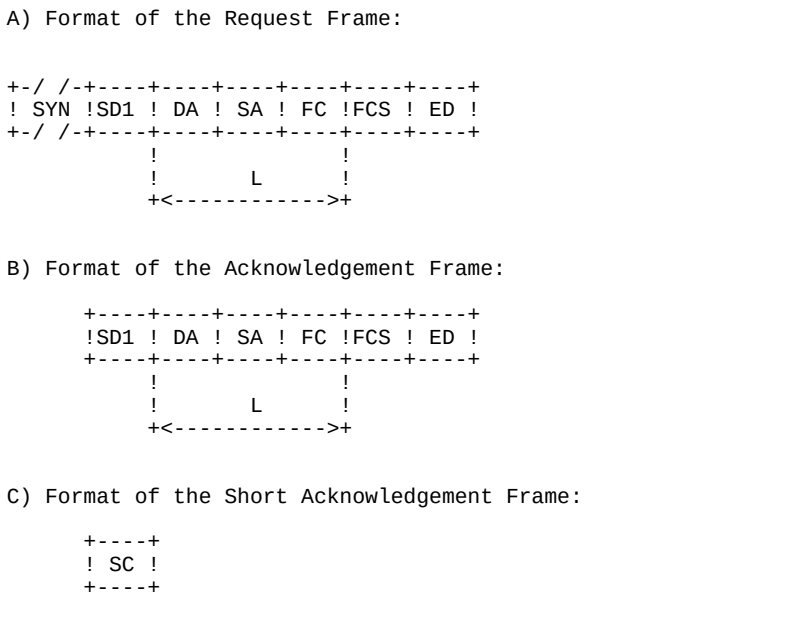
The receiver's bit synchronizing always starts with the falling edge of the start bit, i.e. at the transition from binary "1" to binary "0". The start bit and all consecutive bits are scanned in the middle of the bit time. The start bit shall be binary "0" in the middle of the bit, otherwise the synchronizing fails and is stopped. The synchronizing of the UART character ends with the stop bit being binary "1". If a binary "0" bit is encountered instead of the stop bit, a synchronizing error or UART character error is assumed and reported and the next leading edge of a start bit is waited for.

A maximum deviation of ± 0,3 % of the nominal signalling rate for transmission and receipt is allowed (see Part 2, subclause 4.1.1, Data Signalling Rate).

4.6 Frame Formats

The figures contained in the following clauses do **not** show sequences (request → acknowledgement or response), but frame formats of the same category (Hd = 4, fixed length without/with data field and variable length), i.e. the action frames may be followed by different acknowledgement or response frames (see clause 4.8).

4.6.1 Frames of fixed Length with no Data Field



where:

SYN	Synchronization Period, a minimum of 33 line idle bits
SD1	Start Delimiter, value: 10H
DA	Destination Address
SA	Source Address
FC	Frame Control
FCS	Frame Check Sequence
ED	End Delimiter, value: 16H
L	Information Field Length, fixed number of octets: L = 3
SC	Single Character, value: E5H

Figure 10. Frames of fixed Length with no Data Field

Transmission Rules

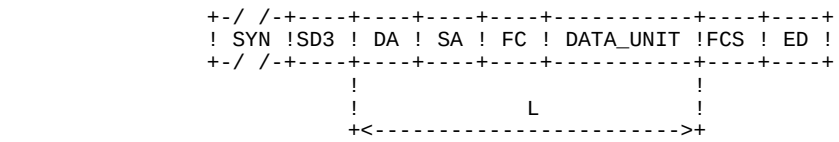
1. Line idle state corresponds to signalling level binary "1".
2. Each action frame shall be preceded by at least 33 line idle bits (Syn Time).
3. No idle states are allowed between a frame's UART characters.
4. The receiver checks:
   per UART character: start bit, stop bit and parity bit (even), per frame: start delimiter, DA, SA, FCS and end delimiter and the SYN Time in the case of an action frame. If the check fails, the whole frame shall be discarded.

SC and SD1 (as well as SD2 and SD3, see subclauses 4.6.2 and 4.6.3) have Hamming distance Hd=4 and are safe against being shifted (see IEC 870-5-1), i.e. the single character SC appears to be a frame with Hd=4.

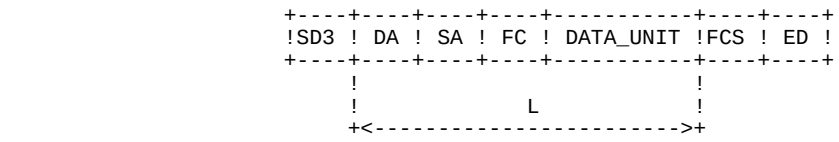
For requests to be acknowledged only (Send Data with Acknowledge), SC is a permissible positive acknowledgement. For requests to be answered (Send and Request Data with Reply), SC is a permissible negative acknowledgement if no data is available (see table 3a, b7=0, Code-No 9).

4.6.2 Frames of fixed Length with Data Field

A) Format of the Send/Request Frame:



B) Format of the Response Frame:



where:

SYN	Synchronization Period, a minimum of 33 line idle bits
SD3	Start Delimiter, value: A2H
DA	Destination Address
SA	Source Address
FC	Frame Control
DATA_UNIT	Data Field, fixed Length (L-3) = 8 octets
FCS	Frame Check Sequence
ED	End Delimiter, value: 16H
L	Information Field Length, fixed number of octets: L = 11

Figure 11. Frames of fixed Length with Data Field

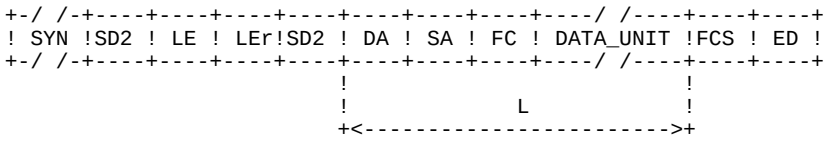
Transmission Rules

The same transmission rules as for frames of fixed length with no data field are valid here (see subclause 4.6.1).

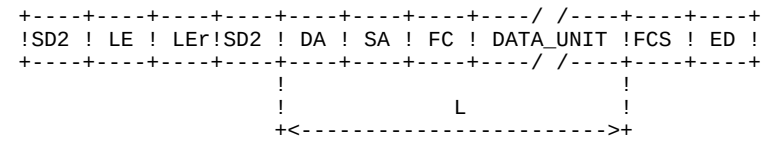
4.6.3 Frames with variable Data Field Length

For a variable number of data octets the length shall also be transmitted in the frame. This length information is contained twice in a fixed frame header at the beginning of the frame. Thus it is protected with Hd = 4 and safe against slip.

A) Format of the Send/Request Frame:



B) Format of the Response Frame:



where:

SYN	Synchronization Period, a minimum of 33 line idle bits
SD2	Start Delimiter, value: 68H
LE	Octet Length, allowed values: 4 to 249
LEr	Octet Length repeated
DA	Destination Address
SA	Source Address
FC	Frame Control
DATA_UNIT	Data Field, variable Length (L-3), max. 246 octets
FCS	Frame Check Sequence
ED	End Delimiter, value: 16H
L	Information Field Length, variable number of octets: L = 4 to 249

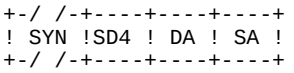
Figure 12. Frames with variable Data Field Length

Transmission Rules

The same transmission rules as for frames of fixed length with no data field are valid (see subclause 4.6.1).

In addition to transmission rule 4, LE shall be identical to LEr, and the information octets shall be counted from the destination address (DA) up to the frame check sequence (FCS) and the result shall be compared with LE.

4.6.4 Token Frame



where:

SYN	Synchronization Period, a minimum of 33 line idle bits
SD4	Start Delimiter, value: DCH
DA	Destination Address
SA	Source Address

Figure 13. Token Frame

Transmission Rules

1. Line idle state corresponds to signalling level binary "1".
2. Each token frame shall be preceded by at least 33 line idle bits (Syn Time).
3. No idle states are permitted between a frame's UART characters.
4. The receiver checks:
   per UART character: start bit, stop bit and parity bit (even),
   per frame: Syn Time, start delimiter and DA/SA. If the check fails, the whole frame shall be discarded.

4.7 Length, Address, Control and Check Octet

4.7.1 Length Octet (LE, LEr)

The two length octets of identical value in the frame header of the variable format contain the number of information octets in the frame body. These comprise: DA, SA, FC and the DATA_UNIT. The value covers the range from 4 to 249, so that a maximum of 246 octets may be transmitted in a frame's DATA_UNIT (see subclause 4.7.5). A value < 4 is not permitted, as a frame contains at least DA, SA, FC and **one** DATA octet. The longest frame contains a total of 255 octets.

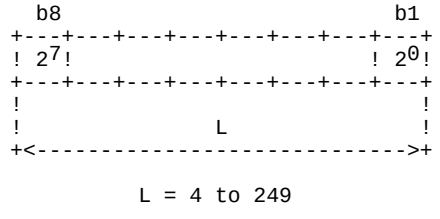


Figure 14. Length Octet Coding

4.7.2 Address Octet (DA/SA)

The two address octets in the frame header (action, acknowledgement and response frames) contain the destination (DA) and source (SA) station address. The token frame consists only of these two address octets after the start delimiter.

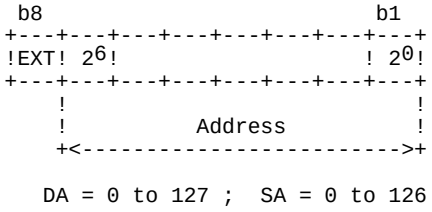


Figure 15. Address Octet Coding

Address 127 (b1 to b7 = 1) is reserved as global address for broadcast and multicast messages (frame to all stations or a group of stations selected by means of a service access point; only permitted in Send Data with No Acknowledge, SDN).

Thus, 127 station addresses (0 to 126) are available for slave and master stations, of which preferably no more than 32 may be occupied by master stations. For non time-critical applications up to 127 master stations are optionally permitted. As at least one master station is required, a maximum of 126 addresses for slave stations is possible.

The action frame's address octets shall be sent back mirrored in the acknowledgement or response frame, i.e. SA of the acknowledgement or response frame contains the destination station address and DA contains the source station address of the action frame.

Address Extension (EXT):

In frames with DATA_UNIT the EXT bit (extension) indicates a destination and/or source address extension (DAE, SAE), which immediately follows the FC octet in the DATA_UNIT. It may be distinguished between access address (Link Service Access Point, LSAP, see subclause 4.7.2.2) and region/segment address. Both address types may also occur simultaneously, as each address extension contains an EXT bit again (see bit b8 in Fig. 17).

The address extensions of the action frame shall be sent back mirrored in the response frame.

EXT = 0 : No address extension in the DATA_UNIT

EXT = 1 : Address extension follows in the DATA_UNIT

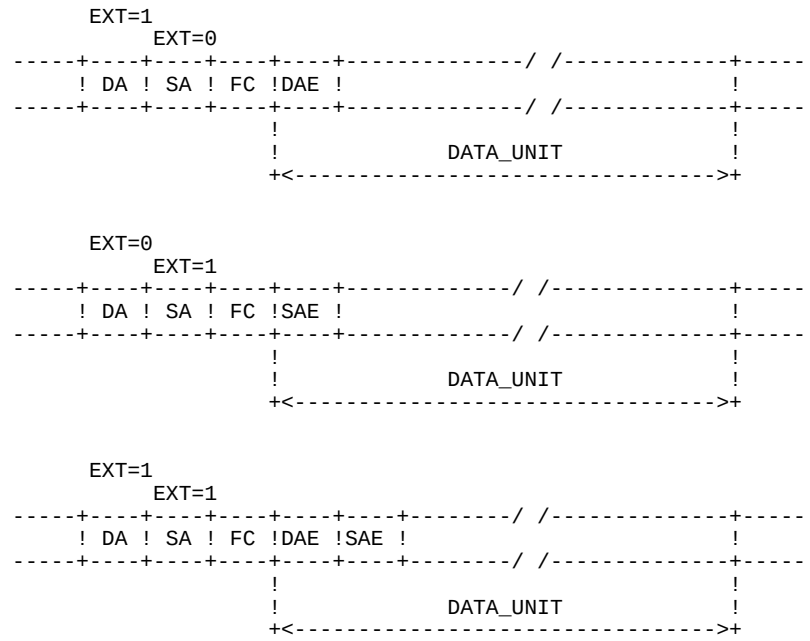
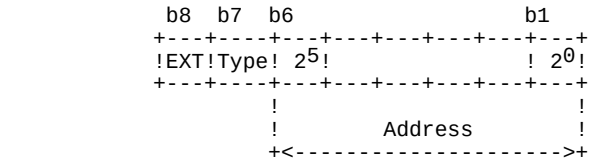


Figure 16. DAE/SAE Octet in the Frame



b7 denote the Type:

0 6 bit Link Service Access Point (LSAP):

DAE = 0 to 63 ; SAE = 0 to 62

1 6 bit Region/Segment Address for Realization of hierarchical Bus Systems with Bridges, Range of Values is to be defined.

b8 (EXT) denotes an additional address extension:

0 No additional address extension octet
1 One additional address extension octet follows **immediately** with the same structure. The following order is valid:
First Octet : Region/Segment Address with b7=1, b8=1
Second Octet: LSAP with b7=0, b8=0

Figure 17. Address Extension Octet

4.7.2.1 Address Check

A receiver checks the destination address information in a frame addressed to itself by following rules with TS:

- Is there no Region/Segment address in the frame existent (DA[b8=0] or DAE[b7=0]), it merely shall check the DA on equality.
- Is there a Region/Segment address in the frame existent (DAE[b7=1]), it shall check the DA and the DAE on equality. Does TS of the receiver not contain a Region/Segment address then the frame count not addressed to the receiver.

4.7.2.2 Link Service Access Point (LSAP)

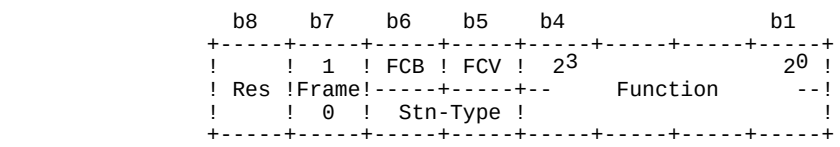
At the FDL user - FDL interface (see chapter 5) a data transmission service (or several thereof) is processed via a Link Service Access Point (LSAP). Several LSAPs at the same time are permitted in master and slave stations. In this case the related LSAP shall be transmitted together with the message.

The address extensions DAE and SAE are available for the transmission of LSAPs. The Source Service Access Point (SSAP), which represents the access address of the local user to the FDL, is transmitted in the SAE octet. The Destination Service Access Point (DSAP), which represents one or all access addresses of the remote user to the FDL, is transmitted in the DAE octet. SSAP values from 0 to 62 and DSAP values from 0 to 63 may be chosen. The DSAP value 63 (DAE b1 to b6 = 1) represents the global access address. This DSAP is only allowed for the send data services "SDA" and "SDN" (see Part 3, clause 4).

If the transmission of LSAPs is omitted for reasons of frame efficiency, the data transmission services shall be processed via the **Default LSAP**. In this case all action frames are transmitted without SAE. All correctly received acknowledgement or response frames without DAE are assigned to this default LSAP. At the FDL user - FDL interface (see Part 3, clause 4.1) the default LSAP is mandatory and is addressed with the value NIL.

4.7.3 Control Octet (FC)

The control octet in the frame header indicates the frame type, such as action frame (request or send/request frame) and acknowledgement or response frame. In addition the control octet contains the function and the control information, which prevents loss and multiplication of messages, or the station type with the FDL state.



Res: Reserved (the sender shall be set binary "0", the receiver does not have to interpret)

Frame Type: 1 Request, Send/Request Frame
0 Acknowledgement, Response Frame

b7 = 1:

FCB Frame Count Bit: 0/1, alternating

FCV Frame Count Bit valid:
0 alternating Function of FCB is invalid
1 alternating Function of FCB is valid

b7 = 0:

Stn-Type: Station Type and FDL Status

b6	b5	<-- Bit Position
0	0	Slave Station
0	1	Master Station not ready to enter logical token ring
1	0	Master Station ready to enter logical token ring
1	1	Master Station in logical token ring

Function: see Table 3a

Figure 18. FC Octet Coding

Table 3a. Transmission Function Codes

Code No	Function	Format in Clause
	Frame Type b7 = 1	
0,1,2	Reserved	
3	Send Data with Acknowledge low	4.6.2/3A
4	Send Data with No Acknow. low	- " -
5	Send Data with Acknowledge high	- " -
6	Send Data with No Acknow. high	- " -
7	Reserved (Req. Diagnosis Data)	
8	Reserved	
9	Request FDL Status with Reply	4.6.1A
10,11	Reserved	
12	Send and Request Data low	4.6.1/2/3A
13	Send and Request Data high	- " -
14	Request Ident with Reply	4.6.1A
15	Request LSAP Status with Reply	4.6.1/3A
	(Code No 14 and 15: FMA1/2)	
	Frame Type b7 = 0	
0	ACKnowledgement positive (OK)	4.6.1B*)
1	ACK negative	
	FDL/FMA1/2 User Error (UE)	4.6.1B
2	ACK negative	
	no Resource for Send Data	
	(& no Response FDL Data) (RR)	- " -
3	ACK negative	
	no Service activated (RS)	- " -
4 to 7	Reserved	
8	Response FDL/FMA1/2 Data low	
	(& Send Data ok) (DL)	4.6.2B, 4.6.3B
9	ACK negative	
	no Response FDL/FMA1/2 Data,	
	(& Send Data ok) (NR)	4.6.1B*)
10	Response FDL Data high,	
	(& Send Data ok) (DH)	4.6.2B, 4.6.3B
11	Reserved	
12	Response FDL Data low,	
	no Resource for Send Data (RDL)	4.6.2B, 4.6.3B
13	Response FDL Data high,	
	no Resource for Send Data (RDH)	- " -
14,15	Reserved	

b4 b1
Code No 0 = 0 0 0 0
Code No 15 = 1 1 1 1

( ) Value of the L/M_status Parameter of the Service Primitives (see Part 3, subclause 4.1.3 and 4.2.3)

*) Frames equivalent to Short Acknowledgement SC = E5H, exception: no SC if FDL Status Request

Frame Count Bit

The Frame Count Bit FCB (b6) prevents the duplication of messages at the responder and the loss at the initiator. However, "Send Data with No Acknowledge" (SDN), "Request FDL Status", "Request Ident" and "Request LSAP Status" are excluded from this.

In order to manage the security sequence, the initiator shall carry a FCB for each responder. When an action frame (request or send/request frame) is transmitted to a responder for the first time or again to a responder currently marked as "non operational", the associated FCB shall be set definitely. The initiator achieves this by an action frame with FCV=0 and FCB=1. The responder shall classify such a frame as first message cycle and store FCB=1 together with the initiator's address (SA and optional SAE[b7=1]) (see table 3b). This message cycle is not repeated by the initiator.

If a responder supports the Region/Segment addressing and the request frame contains a source Region/Segment address (SAE[b7=]) which is unequal to the own Region/Segment address (DAE[b7=1]), then the responder shall store the SAE[b7=1] together with the SA.

In the following action frames to the same responder the initiator shall set FCV=1 and toggle FCB with each new action frame. The responder shall evaluate FCB when receiving an action frame addressed to itself with FCV=1. A FCB changed in comparison with the same initiator's (same SA and optional same SAE[b7=1]) preceding action frame is considered as confirmation of the preceding message cycle's correct completion. If the action frame originates from a different initiator (different SA or optionaldifferent SAE[b7=1]), there is no evaluation of the FCB. In both cases the responder shall store the FCB with the source address (SA and optional SAE[b7=1]) until it receives a new frame addressed to itself.

If an acknowledgement or response frame is missing or corrupted, the FCB shall **not** be changed by the initiator in the retry; this indicates the faulty preceding message cycle. If a responder receives an action frame with FCV=1 and the same FCB as in the same initiator's (same SA and optional same SAE[b7=1]) immediately preceding action frame, a retry is being carried out. As a result the responder shall again transmit the acknowledgement or response frame kept in readiness.

The responder shall keep ready the preceding acknowledgement or response frame for a potential retry, until it gets the above mentioned confirmation, or until it receives a Hd4-faultless frame with changed address (SA or DA or optional SAE[b7=1] or DAE[b7=1]), or a Send Data with No Acknowledge (SDN), or a token frame.

For "Send Data with No Acknowledge", "Request FDL Status", "Request Ident" and "Request LSAP Status", FCV and FCB are = 0; the responder does not analyze FCB.

Table 3b. FCB, FCV in Responder

b6 b5 <-- Bit Position			
FCB	FCV	Condition	Meaning	Action
0	0	DA = TS/127	Request with No Ack Request FDL-Status/ Ident/ LSAP-Status	last Ack or delete Reply
0/1	0/1	DA # TS	Request to other Responder	last Ack or Reply may be deleted
1	0	DA = TS	First Request	FCBM := 1 SAM := SA*) last Ack or delete Reply
0/1	1	DA = TS SA = SAM FCB # FCBM	New Request	last Ack or delete Reply FCBM := FCB have Ack or Reply ready for Retry
0/1	1	DA = TS SA = SAM FCB = FCBM	Request Retry	FCBM := FCB repeat Ack or Reply and keep it ready
0/1	1	DA = TS SA # SAM	New Initiator	FCBM := FCB SAM := SA*) have Ack or Reply ready for Retry
--	--	Token- Frame	---	last Ack or Reply may be deleted

FCBM stored FCB
SAM stored SA

4.7.4 Check Octet (FCS)

The check octet FCS which is required for Hamming distance 4 in a frame always immediately precedes the end delimiter. It is structured as follows:

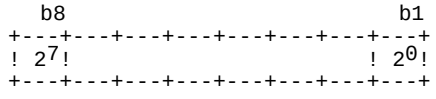


Figure 19. FCS Octet Coding

In frames of fixed length with no data field (see subclause 4.6.1) the check octet shall be calculated from the arithmetic sum of DA, SA and FC without start and end delimiter and without taking the carry-overs into consideration.

In frames of fixed length with data field (see subclause 4.6.2) and in frames with variable data field length (see subclause 4.6.3) the check octet shall additionally include the DATA_UNIT.

4.7.5 Data Field (DATA_UNIT)

The data field consists of an address field and the user data of the FDL/FMA1/2 user. The address field contains from 0 to at most 4 address extension octets (see subclause 4.7.2). The user data without address extension comprises a maximum of 246 octets. The definition and interpretation of the user data (L_sdu) for the data transmission services (see Part 3, clause 4.1) are described in Part 6 of this specification.

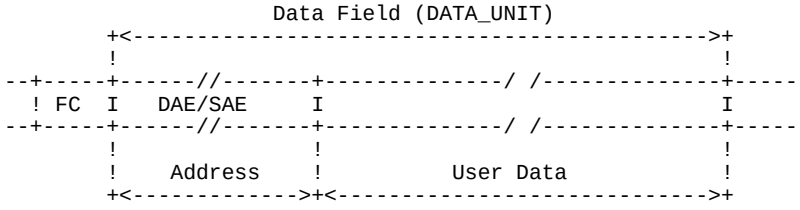
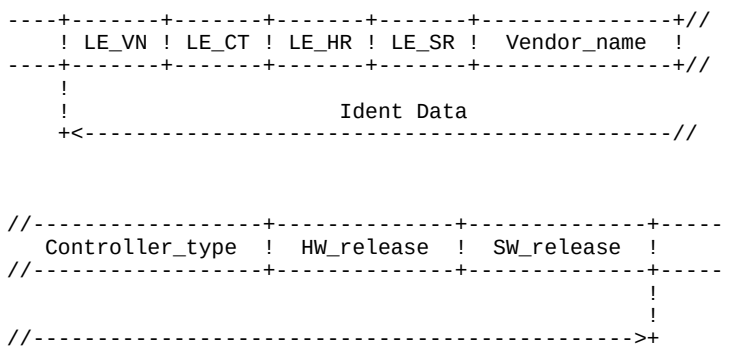


Figure 20. Data Field

The following user data are defined for the remote management services "Request Ident" and "Request LSAP Status" (see Part 3, clause 4.2):

Ident User Data:

The Ident user data contains the station's Ident_List with vendor name (Vendor_name), PROFIBUS controller type (Controller_type) and hardware and software release (HW/SW_release). It comprises a maximum of 200 octets:



LE_VN, LE_CT, LE_HR, LE_SR:

1 octet each. Length of corresponding Data Field in octets;
dual coding, significance as in Fig. 19.

Vendor_name, VN:
Name of Manufacturer as ASCII String (ISO 7 Bit Code, b8=0).

Controller_type, CT:
Hardware Controller Type as ASCII String (ISO 7 Bit Code, b8=0).

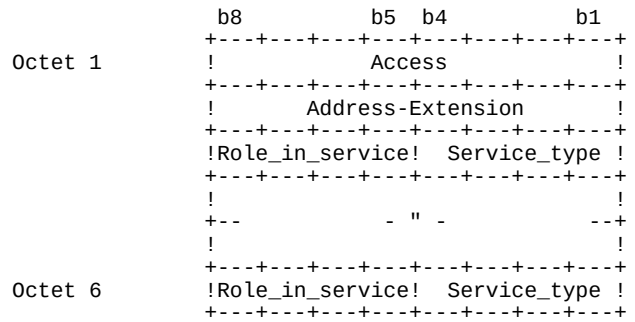
HW_release, HR:
Hardware Release of Controller as ASCII String (ISO 7 Bit Code, b8=0).

SW_release, SR:
Software Release of Controller as ASCII String (ISO 7 Bit Code, b8=0).

Figure 21. Ident User Data

LSAP Status User Data:

The LSAP Status user data contains the configuration of a service access point at the remote station and has the following octets:



Access:

b8 b1 <-- Bit Position

0 1 1 1 1 1 1 = All

EXT 0 0 0 0 0 0 0)
to > 0 to 126 (FDL Address,
EXT 1 1 1 1 1 1 0) Station Address)

EXT = 0: Address Extension invalid
EXT = 1: Address Extension valid

Address Extension:

b8=0, b7=1, b1 to b6: Region/Segment Address
(Significance: see Fig. 17)

Service_type:

b4 b1 <-- Bit Position

0 0 0 0 Send Data with Acknowledge (SDA)
0 0 0 1 Send Data with No Acknowledge (SDN)
0 0 1 1 Send and Request Data with Reply (SRD)
0 1 0 1 Cyclic Send and Request Data
with Reply (CSRD)

Role_in_service:

b8 b5 <-- Bit Position

0 0 0 0 Initiator
0 0 0 1 Responder
0 0 1 0 Both
0 0 1 1 Service not activated

Note: For Service_type SRD and CSRD the Role_in_service is identical for Responder

Figure 22. LSAP Status User Data

4.8 Transmission Procedures

Permissible frame sequences (message cycles) are described in the following. Error sequences, broadcast/multicast messages and "Send Data with No Acknowledge" are excluded.

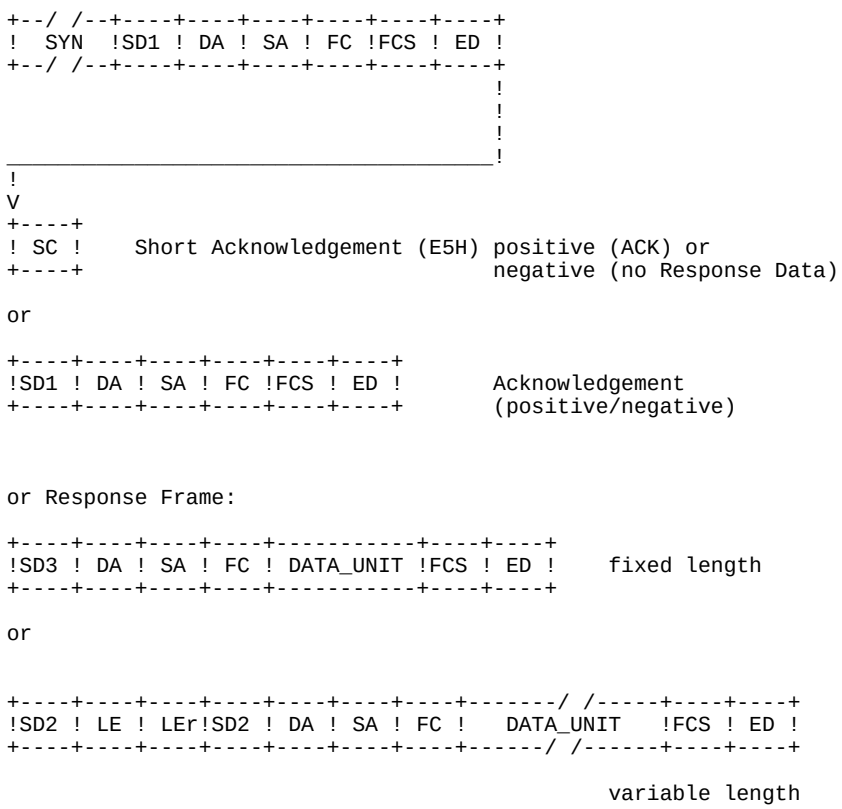


Figure 23. Send/Request Frame of fixed Length with no Data

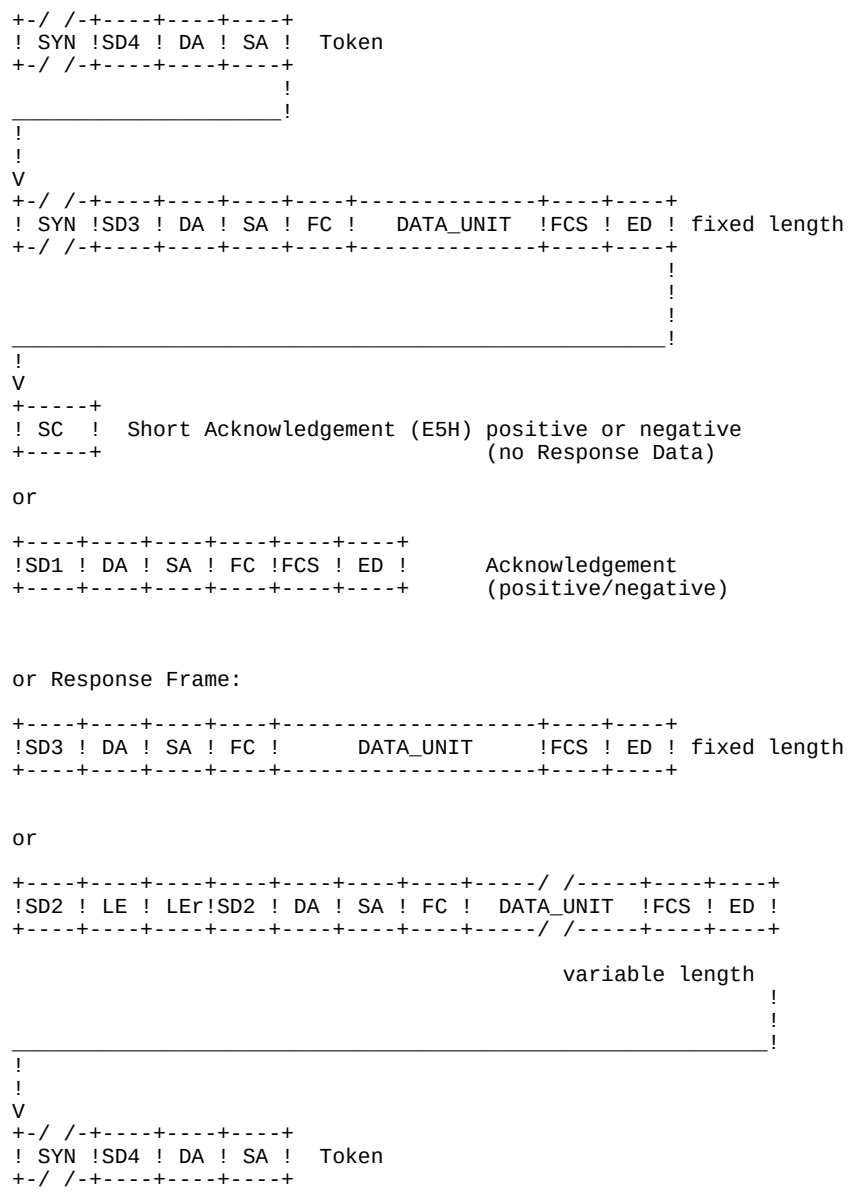


Figure 24. Token Frame and Send/Request Frame of fixed Length with Data including Address Extension

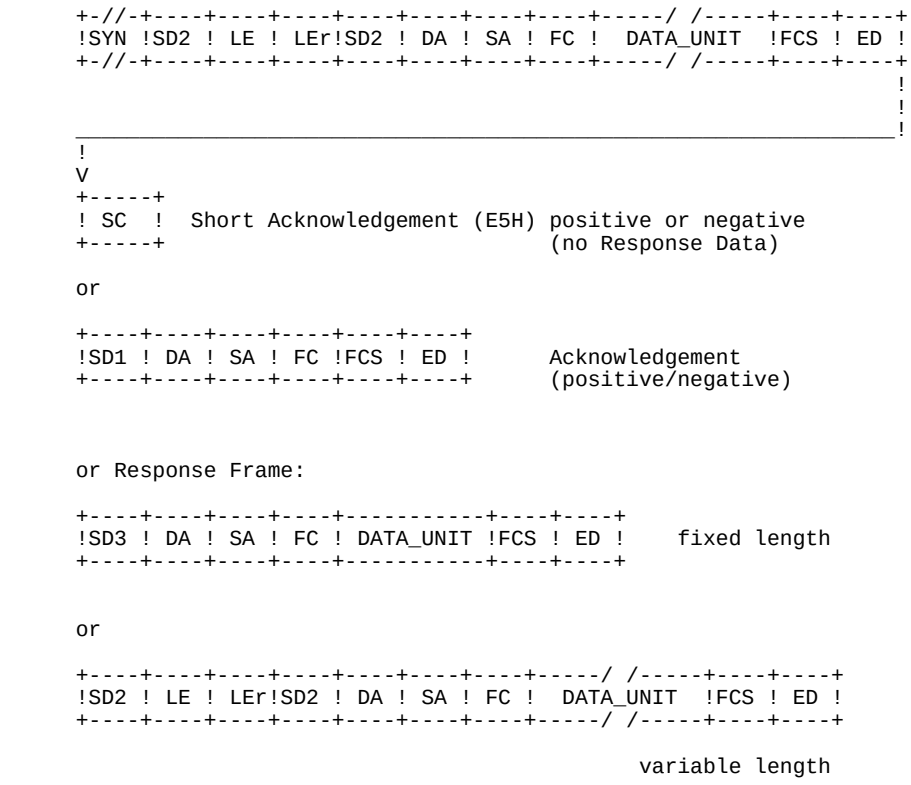


Figure 25. Send/Request Frame with variable Data Field Length

PROFIBUS Specification - Normative Parts
Part 5

Application Layer Service Definition

CONTENTS

Page

1 Scope

This specification specifies the Application Layer protocol, the Application Layer interface, the corresponding Network Management and the mapping onto the Data Link Layer of the bit serial PROFIBUS System according to the PROFIBUS Data Link Layer.

PROFIBUS is intended for automation applications that are close to the process and makes simple bus interfaces with real-time behaviour possible. This specification allows field automation components of different manufacturers to be interconnected in a distributed system and guarantees reliable communication. Such a system is called an "open system". In addition the PROFIBUS protocol makes possible the easy integration into hierarchically higher automation systems (Manufacturing Automation Protocol, MAP). Thus the interconnection expense is minimized.

With the degree of freedom in the specification a flexibility is achieved that allows the implementation of different system configurations and functional structures for different application areas. It is intended to define specific profiles (application-oriented functional standards) for different application areas such as building automation, discrete part manufacturing, process control etc.

2 Normative References and additional Material

ISO 7498/1:1989	Information processing systems; Open System Interconnection; Basic Reference Model
ISO 7498/4:1989	Information processing systems; Open System Interconnection; Basic Reference Model; Part 4: Management Framework
ISO 8824:1987	Information processing systems; Open System Interconnection; Specification of Abstract Syntax Notation One (ASN.1)
ISO 8825:1987	Information Processing Systems; Open System Interconnection; Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)
ISO TR 8509:1987	Information Processing Systems; Open System Inter-connection - Service Conventions
IEEE 754:1985	IEEE Standard for Binary Floating-Point Arithmetic

3 General

3.1 Terms and Apprevations

The multiplicity of technical terms and the need to relate to existing international bus standards make it necessary to limit the technical terms to a well defined set.

Abbreviation	Meaning
.con	confirmation primitive
.ind	indication primitive
.req	request primitive
.res	response primitive
ABO	ABOrt or FMA7 Abort
ACI	Acyclic Control Interval
ACK	ACKnowledged
acyc	acyclic
AD	Additional Detail
ALI	Application Layer Interface
ASIC	Application Specific Integrated Circuit
ASN.1	Abstract Syntax Notation One
ASS	ASSociate service of LLI
BRCT	BRoadCasT communication relationship
C	Conditional
CCI	Cyclic Control Interval
CI	Control Interval
CN	ConNection
CON	CONfirmation
CREF	Communication REFerence
CREL	Communication RELationship
CRL	Communication Relationship List
CRL Header	Header of the Communication Relationship List
CSRD	Cyclic Send and Request Data with reply
"D"	Defined connection
DIN	Deutsches Institut fuer Normung e.V., German standards body
DIS	Draft International Standard
DP-OD	Dynamic list of Program invocations (Object Dictionary)
DS	Disconnected Station local FDL/PHY controller not in logical token ring or disconnected from line
DSAP	Destination Service Access Point
DTA	Data Transfer Acknowledged
DTC	Data Transfer Confirmed
DTU	Data Transfer Unconfirmed
DV-OD	Dynamic list of Variable lists (Object Dictionary)
FC	Function Code
FDL	Fieldbus Data Link, Layer 2
FER	Fieldbus Encoding Rules
FMA	Fieldbus MAnagemant
FMA1/2	Fieldbus Management Layer 1 and 2
FMA7	Fieldbus Management Layer 7
FMS	Fieldbus Message Specification
FR+	Final Response
GAP	Range of station addresses from This Station (TS) to its successor (NS) in the logical token ring, excluding stations above HSA

GAPL	GAP List containing the status of all stations in this station's GAP
HSA	Highest Station Address (FDL Address)
HW	Hardware
"I"	Open Connection at the Requester
ID	Identifier
IDM	Image Data Memory
IEEE	Institute of Electrical and Electronical Engineers, USA
IL	Ident List
IMA	Idle Machine Activated
IND	Indication
INFO	Information Report
INI	Initiate
IS	International Standard
ISO	International Organization for Standardization
IV	Invalid parameters in request
IVID	Invoke ID
kbit/s	kilobit per second (Data Signalling Rate)
L_sdu	Link Service Data Unit
LAS	List of Active Stations
LG	Locally Generated
LL	Live List
LLI	Lower Layer Interface
Loc_add	Local Address
LR	Local Resource not available or not sufficient
LS	Local Service not activated at Service Access Point or local LSAP not activated
LSAP	Link Service Access Point
LSB	Least Significant Bit
M	Mandatory
MAP	Manufacturing Automation Protocol
MMAC	Master-Master connection for ACyclic data transfer
MMS	Manufacturing Message Specification
MSAC	Master-Slave connection for ACyclic data transfer with no Slave initiative
MSAC_SI	Master-Slave connection for ACyclic data transfer with Slave Initiative
MSB	Most Significant Bit
MSCY	Master-Slave connection for CYclic data transfer with no Slave initiative
MSCY_SI	Master-Slave connection for CYclic data transfer with Slave Initiative
MULT	Multicast Communication Relationship
NA	No Acknowledgement/Response
NE	Non Existent
NIL	locally existing value, but not fixed in this specification
NOK	Not OK
"O"	Open connection at the Responder
OD	Object Dictionary
OD-ODES	Object Dictionary Object Description
OSCC	Outstanding Services Counter Client
OSCS	Outstanding Services Counter Server
PDU	Protocol Data Unit
PEE	Poll Entry Enabled
PHY	Physical Layer

PI	Program Invocation
PICS	Protocol Implementation Conformance Statement Proforma
R+	Result(+)
R-	Result(-)
RAC	Receive Acknowledged Request Counter
RC	Reason Code
RCC	Receive Confirmed Request Counter
RDH	Response FDL Data High and no resource for send data
RDL	Response FDL/FMA1/2 Data Low and no resource for send data
Rem_add	Remote address
REQ	Request
RES	Response
ROM	Read Only Memory
RR	no Resource for send data and no Response FDL data (acknowledgement negative)
RS	no Service or no Rem_add activated at Remote Service Access Point (acknowledgement negative)
RSAP	Remote Service Access Point
RSV	Reserve
RTimer	Receive Timer
RVR	ReadValueRem
S	Selection
S-OD	Static list of Objects (Object Dictionary)
SAC	Send Acknowledged request Counter
SAP	Service Access Point
SC	State Conflict
SCC	Send Confirmed request Counter
SDA	Send Data with Acknowledge
SDN	Send Data with NO acknowledge
SI	Slave Initiative
SN	Sign
SRD	Send and Request Data with reply
SSAP	Source Service Access Point
ST-OD	Static list of Types (Object Dictionary)
Std	Standard
STimer	Send Timer
SU	Summer time (standard time / summer time)
SW Release	Software Release
T1	Timer 1
TQUI	Quiet Time, Transmitter fall Time (Line State Uncertain Time) and / or Repeater switch Time;the time a transmitting station must wait after the end of a frame before enabling ist receiver
TRR	Real Rotation Time the time between the last successive receptions of the token by this Master station
TS	This Station
TSDR	Station Delay of Responder the actual time this responder waits before generating a reply frame
TSET	Setup Time the time between an event (e.g. interrupt SYN timer expired) and the necessary reaction (e.g. enabling receiver)
TSL	Slot Time the maximum time a Master station on this PROFIBUS System must wait for a transaction response
TTR	Target Rotation Time the anticipated time for one token rotation on this PROFIBUS System including allowances for high and low priority transactions, errors and GAP maintenance
U	User optional
UE	negative acknowledgement, remote User interface Error
VFD	Virtual Field Device

3.2 Architecture and Placement in the ISO/OSI Layer Model

The PROFIBUS architecture is based on the ISO/OSI model of open communication (ISO 7498). The PROFIBUS specification uses Layer 1 (Physical Layer, PHY), Layer 2 (Data Link Layer, FDL) and Layer 7 (Application Layer). The Layers 3 to 6 are empty to minimize expense and increase efficiency.

The Physical Layer (Layer 1), Data Link Layer (Layer 2) and the corresponding management (FMA1/2) are described in the Data Link Layer of the PROFIBUS Specification. The Application Layer (FMS, LLI) and the corresponding management (FMA 7) are described in this specification.

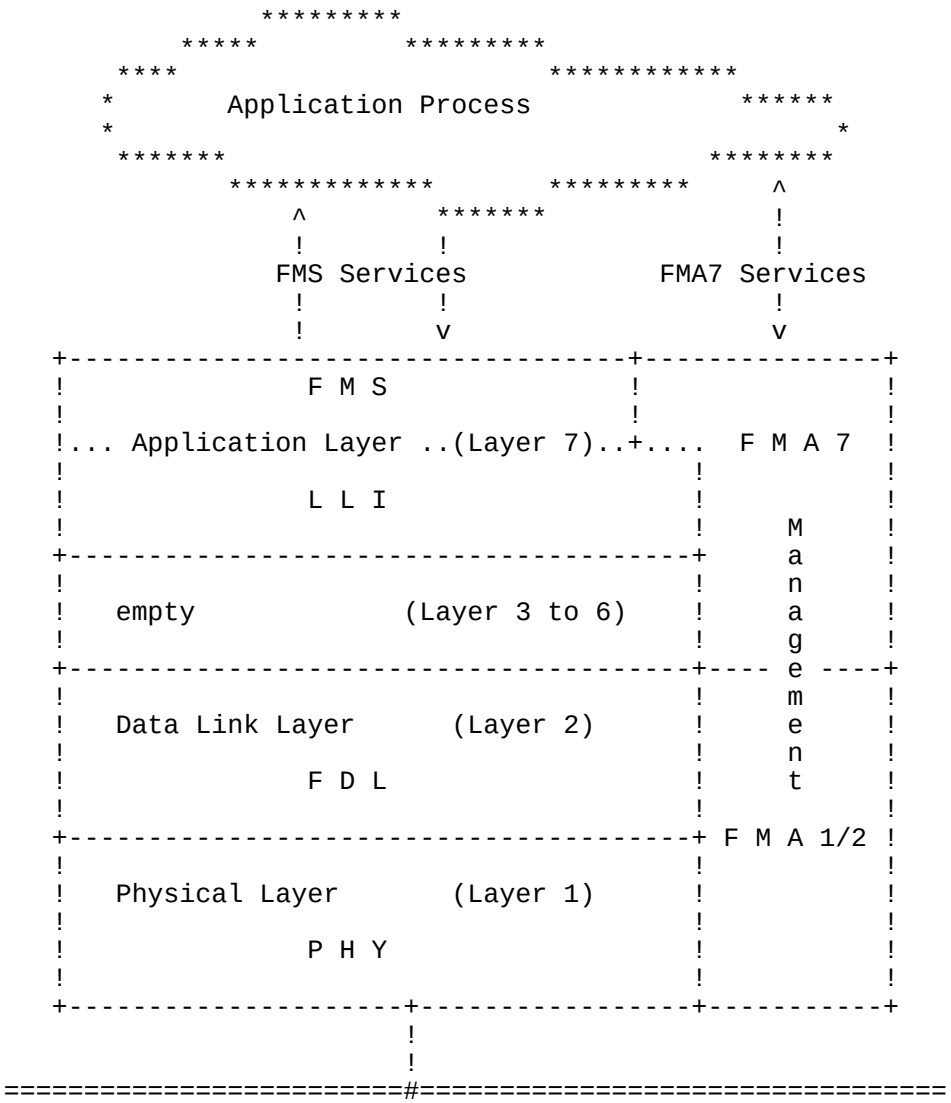


Figure 1. Placement in the ISO/OSI Layer Model

3.2.1 Application Layer

The Application Layer of the PROFIBUS Specification consists of the two entities FMS (Fieldbus Message Specification) and LLI (Lower Layer Interface).

3.2.1.1 Fieldbus Message Specification (FMS)

The FMS describes communication objects, services and associated models from the point of view of the communication partner (server behaviour).

3.2.1.2 Lower Layer Interface (LLI)

The manifold characteristics of the PROFIBUS concept require a particular adaptation between FDL and FMS/FMA7. This adaptation is achieved by means of the LLI. The LLI is an entity of Layer 7.

The main tasks of LLI are:

- mapping of FMS and FMA7 services on to the FDL services
- connection establishment and release
- supervision of the connection
- flow control

3.2.2 Fieldbus Management Layer 7 (FMA7)

FMA7 is based on the system management of ISO DIS 7498-4:1989 and describes objects and management services. The objects are manipulated locally or remotely using management services. The management services are divided into three groups:

context management:
The context management provides services for establishing and releasing a management connection.

configuration management:
The configuration management provides services for the identification of communication components of a station, for loading and reading the Communication Relationship List (CRL) and for accessing variables, counters and the parameters of Layers 1/2.

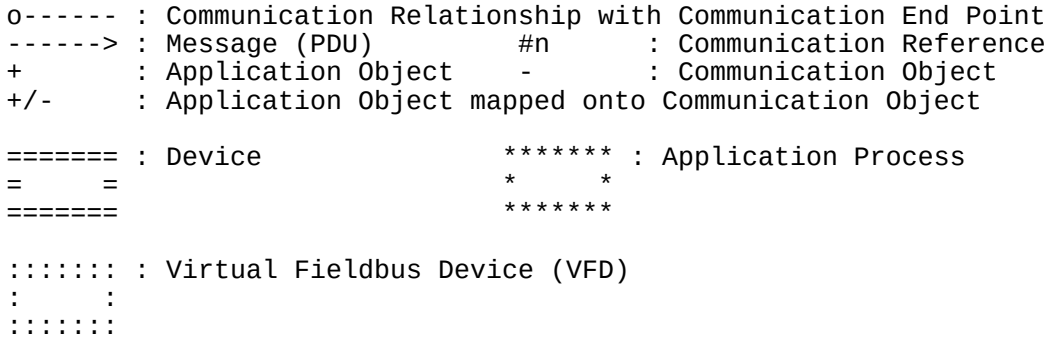
fault management:
The fault management provides services for recognizing and eliminating errors.

3.3 PROFIBUS Communication Model (FMS)

From the communication point of view an application process includes all programs, resources and tasks that are not associated with a communication layer. This includes operating systems, real application processes, application programs and communication drivers (ALI, Application Layer Interface).

The PROFIBUS communication model permits the combination of distributed application processes into a total process via communication relationships (see Fig. 2). The application processes may be distributed on several different devices. One or more application processes may exist in a single device. The application process works with process objects (variables, programs etc.) necessary for the execution of the process. A process object is described by attributes, rules and operations applicable to it. The PROFIBUS communication model supports the object oriented operation of the application process.

Legend to the following figures:



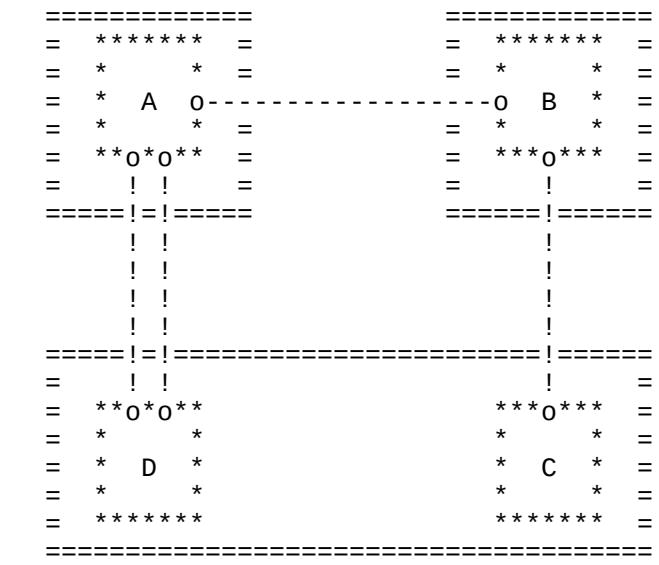


Figure 2. Application Process consisting of Subprocesses A to D

3.3.1 Relationship between Application Process and Communication

An application process has access to the communication using communication end points (see ISO 7498). One or more communication end points are fixed and uniquely assigned to an application process. These end points are addressed by the application process by means of communication references. The communication references are device specific and are not defined by the communication itself. Between two application processes one or more communication relationships may exist, each one having unique communication end points as shown in the following figure.

Figure 3. Assignment of Communication Reference to Application Process

In order to permit an application process to communicate with an application process of an another device, communication objects must be available. Certain communication objects (virtual objects) represent existing process objects, that an application process has made visible and accessible to the communication. Communication objects can be accessed using the FMS services provided.

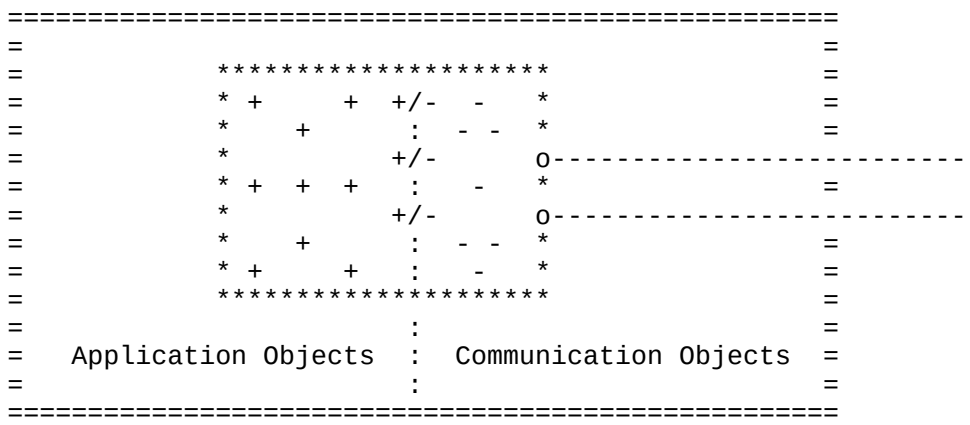


Figure 4. Application Process with Communication Objects

3.3.1.1 Fieldbus Message Specification (FMS) Services

FMS services allow an application process to use the server functionality of a remote application process. The FMS services are transmitted to the communication partner using messages (Protocol Data Units, PDUs). The individual FMS services are elements of communication models that determine the sequence of operations and the rules when using a service. Within a communication model, FMS services act upon communication objects. Certain services use or act upon certain communication objects only.

There are services which are confirmed explicitly by the remote application process after execution (confirmed services) and others where the execution is not confirmed by the application process of the communication partner (unconfirmed services). The parameters required when using a service shall be provided by the application process. The required parameters are defined in a formal, logical description in a tabular way. The name of the parameter is stated in the first column. The further columns are for the specific service primitives. The relationship of a parameter to a service primitive is given by the entry in the row of the parameter and the column of the service primitive.

The following types of relationships are possible:
- The parameter is mandatory.
- The parameter is a user option; it may be used or omitted.
- The parameter may be selected from a set of several parameters.
- The existence of a parameter depends on another parameter.

The parameters may be structured. The name of a subparameter of a structured parameter is shown indented by 2 characters.

The information as to whether an optional parameter is used and which parameter of a set is selected in a concrete case, is not contained explicitly in this representation. However, this information shall be transferred at a concrete interface.

The parameter of a service request and of the service indication is called argument. The service acknowledgements (Res and Con) contain either the parameter Result(+) for the acknowledgement in case of success or Result(-) in case of error. These parameters are normally structured further.

Table 1. Parameters of Service Primitives

Parameter Name	.req	.ind	.res	.con
Argument	M	M=		
Request Parameter1	M	M=		
Parameter_A	S	S=		
Parameter_B	S	S=		
Request Parameter2	U	U=		
Result(+)			S	S=
Acknowledge Parameter1			M	M=
Acknowledge Parameter2			C	C=
Result(-)			S	S=
Error			M	M=

```
.req:  request service primitive
.ind:  indication service primitive
.res:  response service primitive
.con:  confirmation service primitive
   M:  parameter is Mandatory for the primitive
   U:  parameter is a user option; may be provided or omitted
   S:  parameter is a Selection from a collection of two or
       more possible parameters
   C:  parameter is Conditional upon another parameter
```

The code "=" after the code M, U, S or C indicates that the parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table. (For instance, "M=" in the indication service primitive column and "M" in the request service primitive column means that the parameter in the indication primitive is semantically equivalent to that in the request primitive.)

In the example above the argument as the parameter of the request is structured further. It consists of request parameter1 and request parameter2. The request parameter1 may either be parameter_A or parameter_B.

The Result(+) parameter is used for the acknowledgement in the case of success. This parameter consists either of the acknowledge parameter1 or both the acknowledge parameter1 and the acknowledge parameter2. In case of failure, the parameter Result(-) is selected which consists of the parameter error.

3.3.1.2 Virtual Fieldbus Device (VFD) Model

The Virtual Field Device (VFD) model uniquely represents that part of a real application process that is visible and accessible to the communication. The VFD model defines the communication behaviour of that part of the application process (see the folowing figure). The VFD model is based on the VFD object. The VFD object contains all explicit and implicit communication objects and their descriptions. The object descriptions are stored in an Object Dictionary (OD). A VFD object contains exactly one Object Dictionary and is assigned to exactly one application process. A real device may contain several VFD objects, each addressed by its communication end points. The object description for the communication object VFD is defined by the PROFIBUS Specification.

Furthermore the execution of all FMS services is defined by the VFD model.

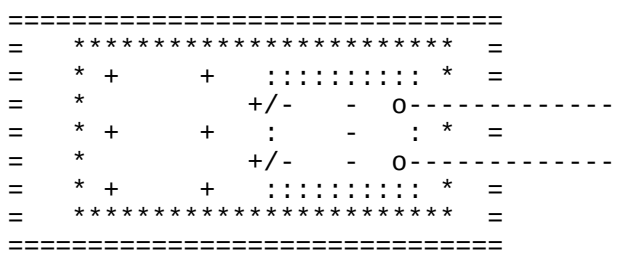


Figure 5. Assignment of VFD to Application Process

3.3.2 Relationship between Client and Server

A client, in terms of communication, is an application process which, with regard to a service, uses the functionality of a remote application process. The

server is the application process, which with regard to a service, provides the functionality of its VFD to the client (see next figure). An application process may therefore on principle be both client and server. FMS services are provided to submit requests. Requests are transmitted over the given communication relationship to the communication partner using a message (PDU).

There is a distinction between confirmed and unconfirmed services. In the case of confirmed services, the client issues the request and the server confirms the execution of the request with an acknowledgement (confirmation).

Unconfirmed services are initiated by the server. In the case of confirmed services, the requests issued on a communication relationship shall have an identification (Invoke-ID) assigned by the application process. This request identification allows unique assignment of the issued request to the corresponding acknowledgement. The execution of the services is described in detail with service primitives and service sequences.

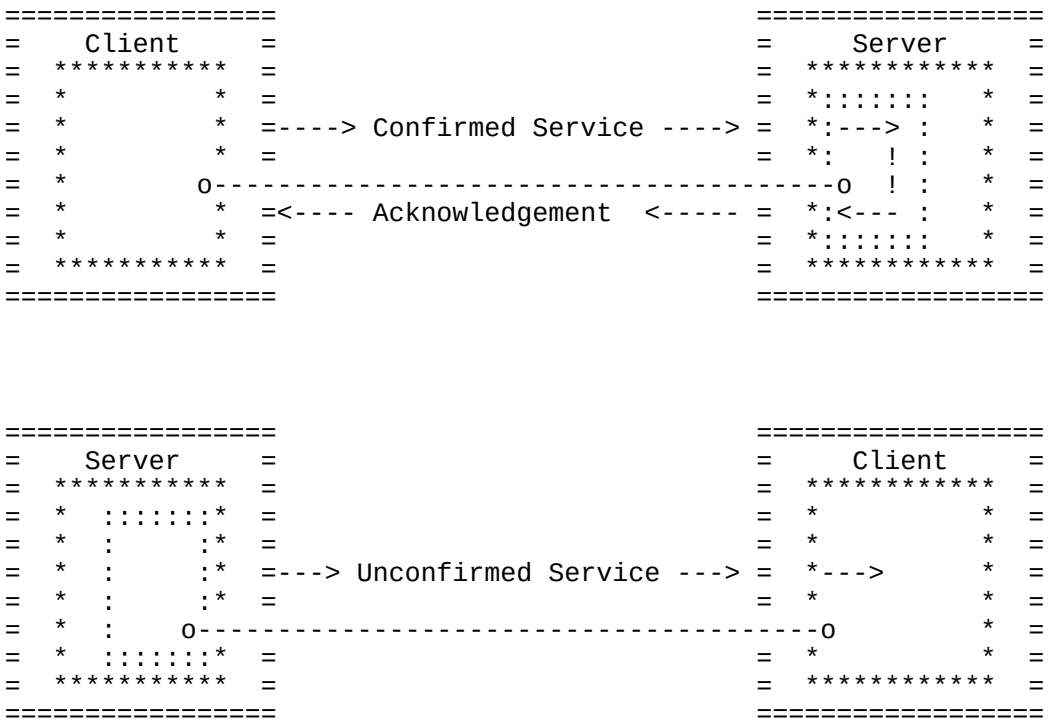


Figure 6. Relationship between Client and Server

3.3.2.1 Service Primitives

The issuing of a request (confirmed service) is described at the client by the service primitive request (.req) and the receipt of the server's acknowledgement with the service primitive confirmation (.con). At the server the receipt of the request is described by the service primitive indication (.ind) and the return of the request acknowledgement with the service primitive response (.res).

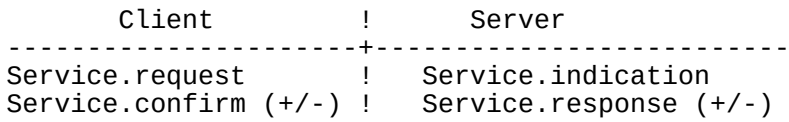


Figure 7. Confirmed Service

Service.request (.req):
With this service primitive the client delivers a service request to the communication.

Service.indication (.ind):
The server shall execute the request upon receipt of this service primitive from the communication.

Service.response (.res):
With this service primitive the server delivers the acknowledgement of the request to the communication.

Service.confirm (.con):
With this service primitive the communication delivers the server's acknowledgement of the request to the client.
An unconfirmed service is initiated by the server using the service primitive request.
The receipt at the client is notified with the service primitive indication.

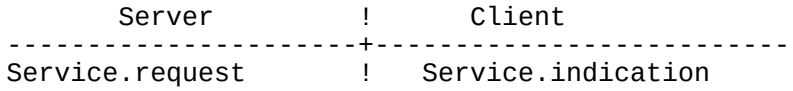


Figure 8. Unconfirmed Service

Service.request (.req):
With this service primitive the server initiates an unconfirmed service.

Service.indication (.ind):
The receipt of an unconfirmed service is notified with this service primitive.

3.3.2.2 Service Sequences

The timing of the service primitives between client and server is shown with service sequences.

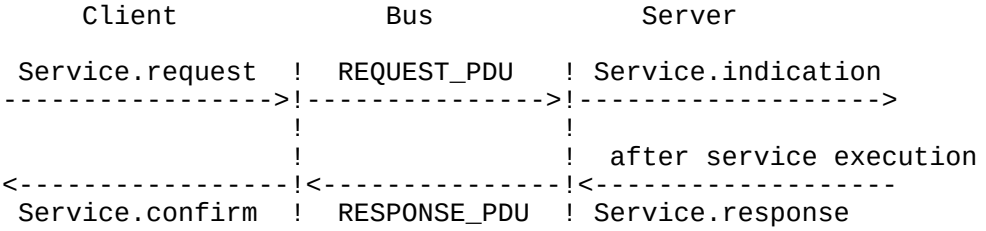


Figure 9. Confirmed Service Sequence

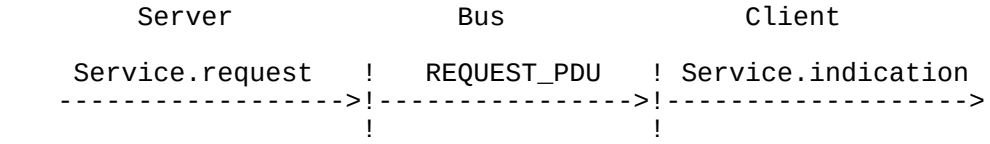


Figure 10. Unconfirmed Service Sequence

3.3.2.3 Parallel Services

Parallel services on a communication relationship exist if several requests in a server from exactly one client are present on a connection at the same time, or if a client has initiated several requests to exactly one server over a connection at the same time that have not yet been executed. The individual requests are distinguished from each other by the request identification (Invoke-ID).

Figure 11. Client - Server / parallel (confirmed) Services

3.3.2.4 Mutual Services

Mutual (confirmed) services exist if an application process is client as well as server for a communication relationship at the same time. That means that this application process has given a request and has received but not yet executed a request see the following figure.

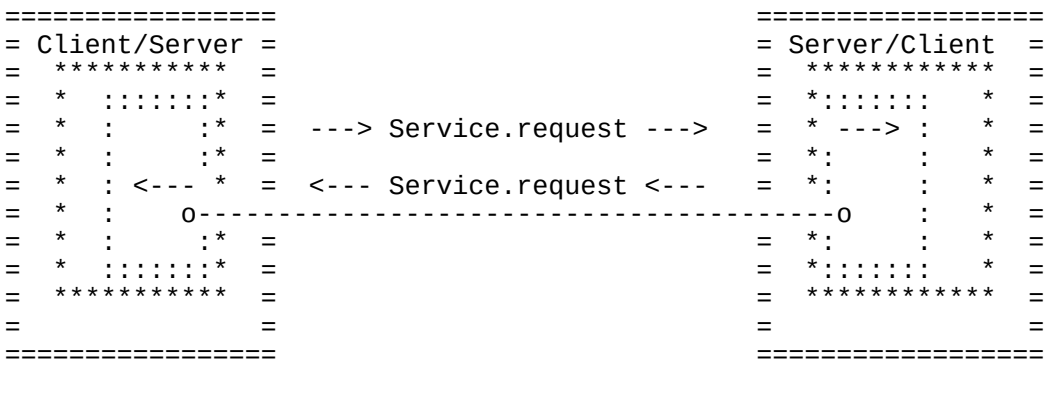


Figure 12. Client - Server / Mutual (confirmed) Services

3.3.3 Communication Relationships between Applications

Each communication relationship is configured, irrespective of the time of its use. The configuration is stored in each station in its Communication Relationship List (CRL). An application process identifies the communication relationship using a local communication reference (application specific, symbol or number). The communication relationships are divided into the following three types:

- one-to-one
- one-to-many
- one-to-all

A further attribute of a communication relationship is the authorization of a station to access the bus. Herein it is distinguished between master - master and master - slave communication relationships. Additional distinguishing features are the attributes "connection-oriented" and "connectionless" as well as the connection types "cyclic" and "acyclic".

The Application Layer of the PROFIBUS Specification uses the services and priorities of the PROFIBUS Data Link Layer (FDL) according to the selected communication relationship. The FDL provides two priorities for the transmission of messages.

3.3.3.1 Communication Relationship between Master and Master as well as Master and Slave

It is possible for a master to access the bus actively, that is to transmit messages independently. Slaves only have the cap-ability of a passive bus access, that is messages may only be transmitted on request of a master. If several masters exist, rules for the authorization of bus access are necessary. This is achieved in a PROFIBUS System by passing the bus access right (token) from master to master. The bus access right shall be passed on by the master no later than upon expiration of the token hold time.

A master is able to poll slaves as long as it has the active bus access; that means that the slaves receive the passive bus access one after another (see PROFIBUS Data Link Layer).

The PROFIBUS Specification distinguishes between master-master and master-slave communication relationships.

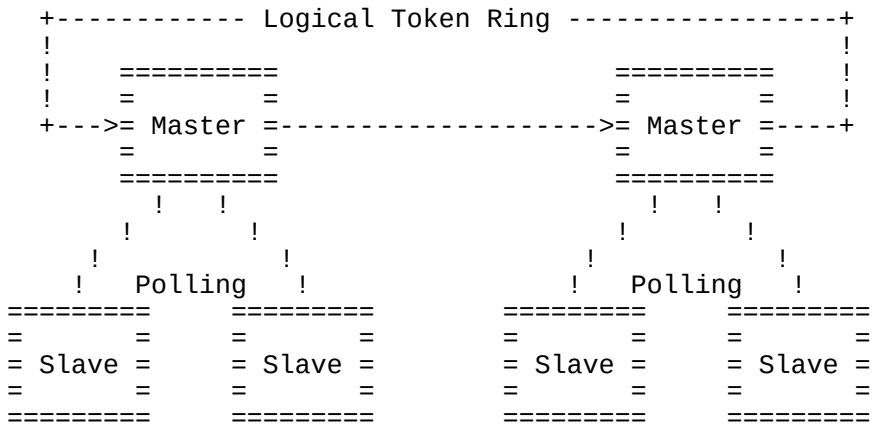


Figure 13. Hybrid Medium Access Control in PROFIBUS

3.3.3.2 Types of Communication Relationships

In a one-to-one communication relationship an application process communicates with exactly one remote communication process. This is realized in PROFIBUS by connection-oriented communication relationships.

In a one-to-many relationship (multicast) one application process communicates concurrently with the application processes of a group of stations. Exactly one application process is addressed per station.

In the case of a one-to-all relationship (broadcast), one application process communicates concurrently with the application processes of all stations. Exactly one application process is addressed per station.

A one-to-many or one-to-all communication relationship does not permit acknowledgement of the execution of a request. One-to-many and one-to-all communication relationships are realized in PROFIBUS by connectionless communication relationships.

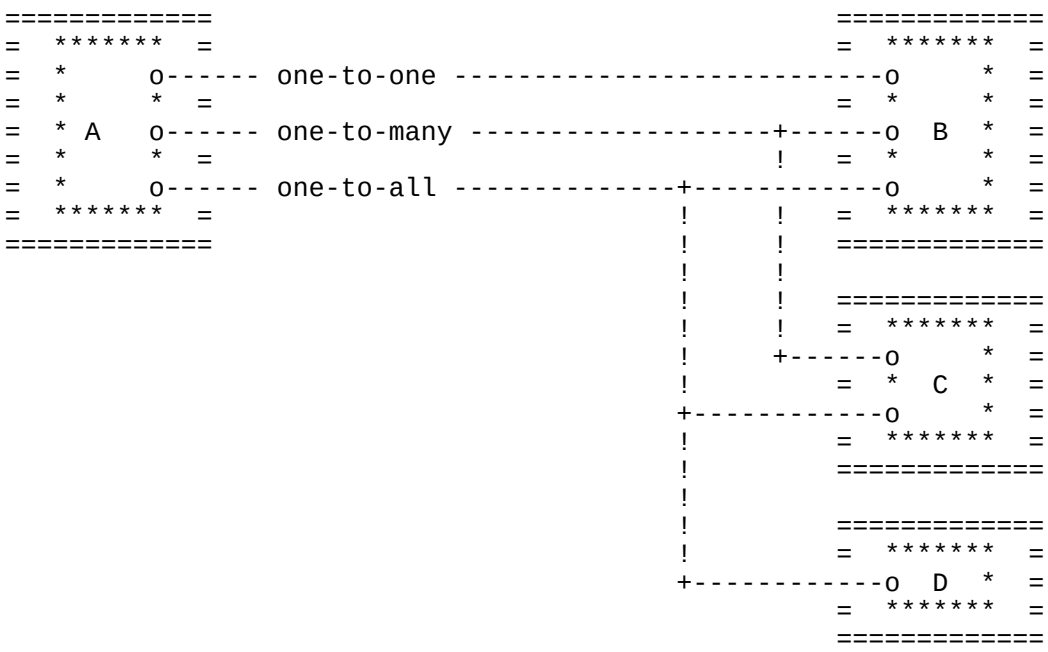


Figure 14. Example of Communication Relationships

3.3.3.2.1 Connection-oriented Communication Relationships

In a connection-oriented communication relationship a logical one-to-one connection is established between two communication partners. For connection-oriented communication one basically distinguishes between
- connection establishment phase
- data transfer phase and
- connection release phase

Exchanging data on a connection shall only be possible after a successful connection establishment. In the connection establishment phase a request to establish the connection is announced at the remote application process (Initiate service). The request to establish the connection contains the information about

the services which will be used in the data transfer phase, the maximum message size and the desired connection type as well as further options of the connection.

If the remote application process accepts the request to establish the connection, it transmits a confirmation of this request to the requester.

After that, both application processes are in the data transfer phase and are able to communicate with each other according to the agreed rules (context). The availability of an established connection may be controlled in the LLI.

A connection is dissolved by its release (Abort service). Thereafter data transfer is only possible when the connection has been established again.

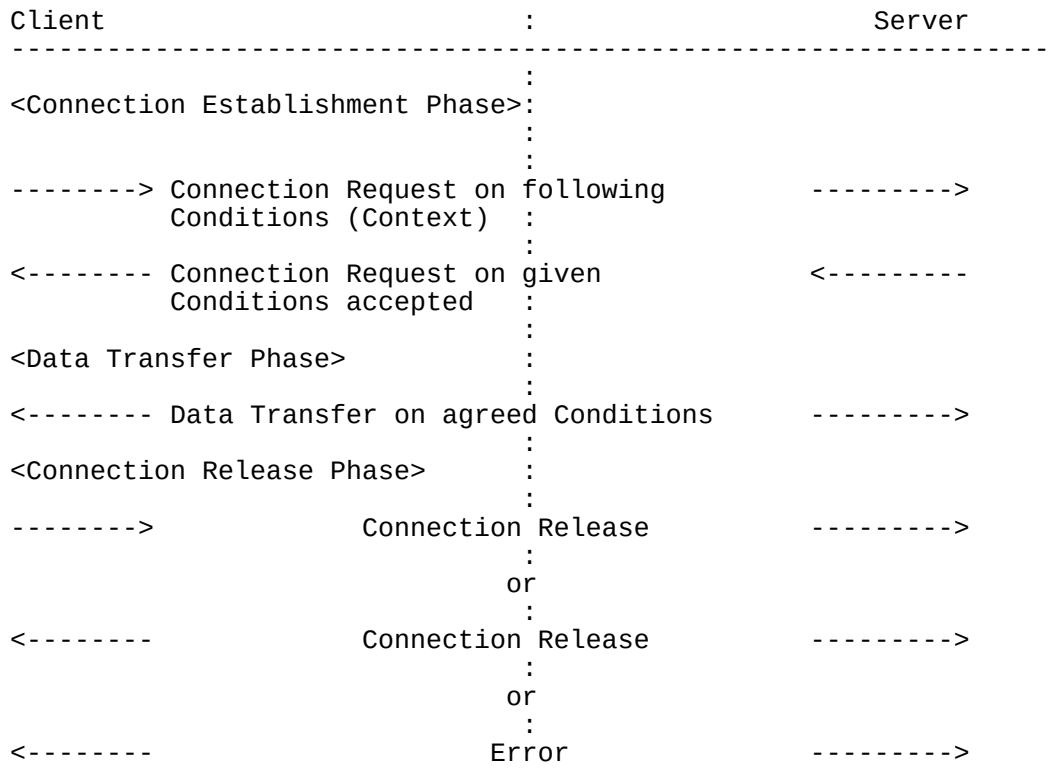


Figure 15. Connection-oriented Communication Relationship

The PROFIBUS Specification defines the following connection types:

communication relationships between master and slave
- connection for cyclic data transfer without slave initiative
- connection for cyclic data transfer with slave initiative
- connection for acyclic data transfer without slave initiative
- connection for acyclic data transfer with slave initiative

communication relationships between master and master

- connection for acyclic data transfer

In a connection for cyclic data transfer confirmed FMS services are executed cyclically by the communication on request of the master in the data transfer phase. Only the FMS services Read and Write are permitted.

In addition the master can initiate unconfirmed FMS services with high or low priority. Unconfirmed FMS services are only executed once per service request; this also applies to a connection for cyclic data transfer.

On a connection for cyclic data transfer with slave initiative the slave may also initiate unconfirmed FMS services with high or low priority.

If a connection for cyclic data transfer with or without slave initiative is required between two masters, one of them shall behave as a slave as far as Layer 7 service sequences are concerned.

In a connection for acyclic data transfer, confirmed and unconfirmed services are only executed once. Services are always initiated by the master (in case of a master - master communication relationship by both).

In a connection for acyclic data transfer with slave initiative, the slave is in addition able to initiate unconfirmed FMS services with selectable priority.

3.3.3.2.2 Connectionless Communication Relationships

A connectionless communication relationship is either a one-to-many (multicast) or a one-to-all (broadcast) communication relationship. For connectionless communication relationships neither a connection establishment nor a connection release is performed. They are always in the data transfer phase. It is not possible for the communication to monitor the connection in this case. The execution of the request cannot be acknowledged by the remote application processes, i.e., only unconfirmed services are permissible.

3.3.3.3 Communication Relationship List (CRL)

The Communication Relationship List (CRL) of a station contains the description of all communication relationships of this station to other stations independent of the time of use (see figure below). The CRL is prepared individually for each PROFIBUS station when configuring the network and loaded locally or remotely using the management services.

Information about the type of communication relationship, the connection type, the context, the addressing and the associated VFD are stored for each communication relationship in the CRL.

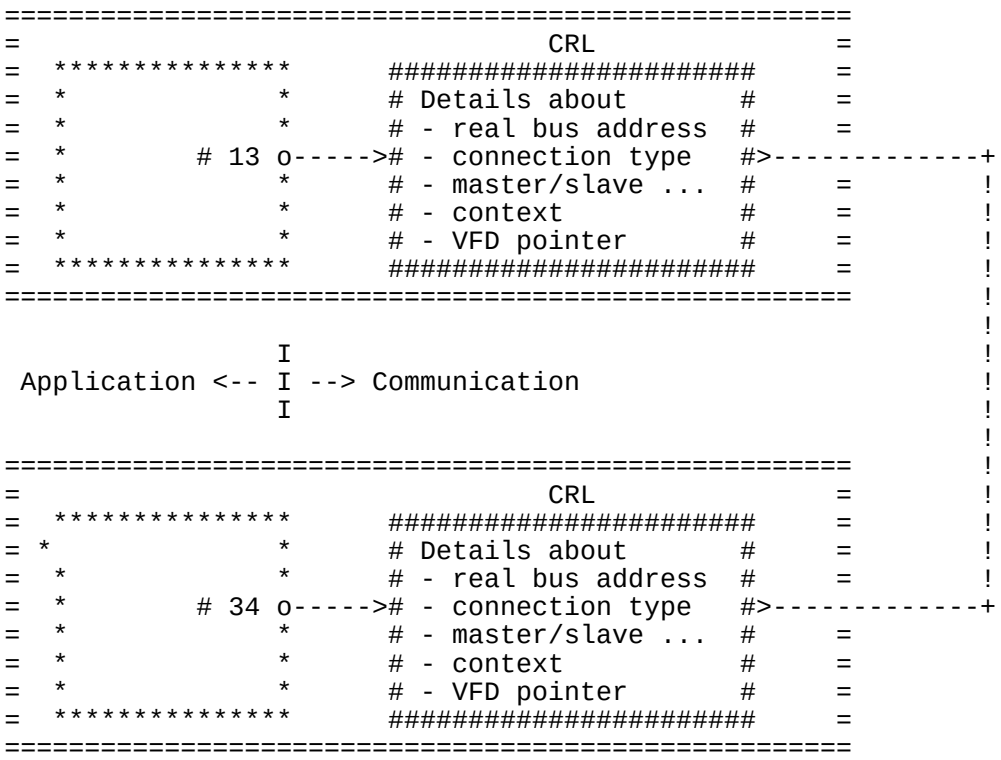


Figure 16. Example of a Communication Relationship List (CRL)

3.3.4 Communication Objects

The PROFIBUS Specification provides a multitude of communication objects to the remote application process. Object specific services act on each communication object. If a service operates on a communication object, the behaviour of the communication object is defined by rules (PROFIBUS models).

The PROFIBUS Specification distinguishes between explicit and implicit communication objects.

The object description of implicit communication objects is defined by the PROFIBUS Specification. Thus the communication may neither create, read, alter nor delete the description of implicit communication objects.

Explicit communication objects are assigned to an object description. The object descriptions of all explicit communication objects are listed in the Object Dictionary of a station.

3.3.4.1 Static and dynamic Communication Objects

Explicit communication objects are divided into two groups:

static communication objects:

simple variable, array, record, domain, event

dynamic communication objects:

program invocation, variable list

Static communication objects shall be registered in the static part of the Ob-

ject Dictionary. In the field area, static objects normally do not change very often or only in certain operational modes. The definition of static communication objects may be performed in one of the following operational phases:

- during start-up or configuration phase

- on line

Typically static communication objects are registered in the Object Dictionary during the projection phase of the bus system.

The dynamic communication objects shall be entered in the dynamic part of the Object Dictionary. They may be predefined or created and deleted dynamically using FMS services.

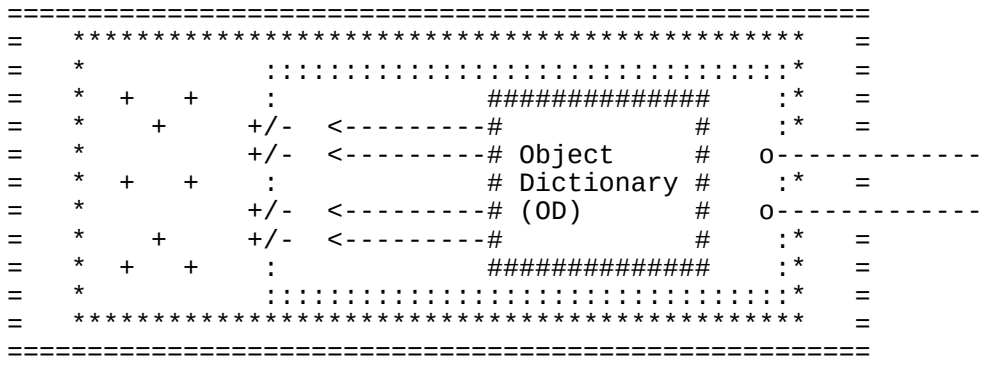


Figure 17. Assignment of Object Dictionary to VFD

3.3.4.2 Object Addressing

Each communication object shall be able to be addressed. The PROFIBUS Specification defines the following addressing schemes:
- physical
- implicit
- logical addressing or addressing by names

The addressing scheme depends on the object. Explicit objects may be addressed logically or by name. In the case of explicit objects logical addressing is mandatory, whereas addressing by name is optional.

In the case of logical addressing, each communication object shall be addressed by use of a logical address (index) that shall be read out of the Object Dictionary (see figure below). The addressing of the process object is done in the application process itself. The assignment between logical address (communication) and real address (process object) is defined by local rules. The communication partner shall read this assignment from the Object Dictionary if it is not known from the configuration.

In the case of addressing by names, the names shall be unique, at least within the following groups of objects:
- domain
- program invocation
- simple variable, array, record
- variable list
- event

The structure of the names and their use is defined by the application and by profiles. In the case of physical addressing, the address is specific to the device and thus not defined by the PROFIBUS Specification. Implicit addressing applies to communication objects that are important for the administration of the communication itself (VFD, transaction object).

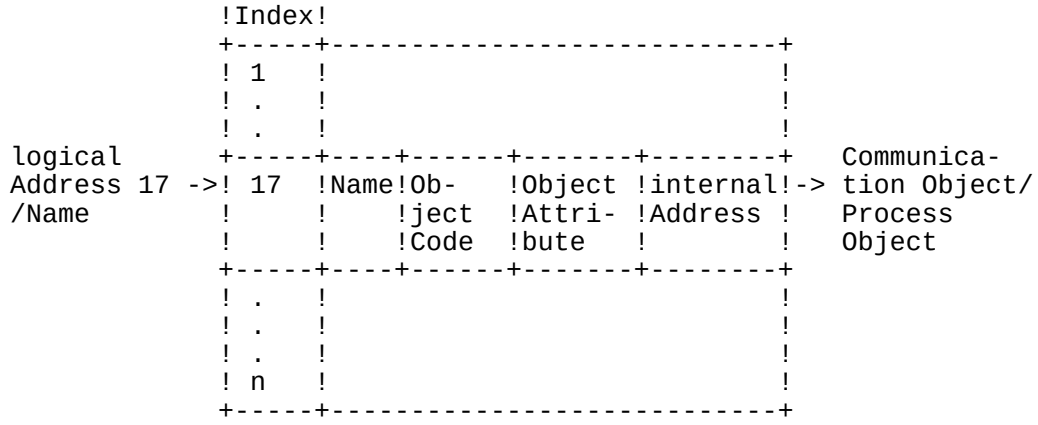


Figure 18. Object Dictionary (Principle)

3.3.5 Object Description Model of Communication

A remote application process has to know the object description before it is able to operate on the communication object. This information may be announced during configuration. If an application process does not know the object description when accessing an object, the description of the object shall be read at the communication partner.

The station where the object really exists holds the definition of the object description (Source-OD). The Source-OD contains the description of all communication objects of a station.

The Object Dictionary is designed specifically for each VFD. The structure defined in this specification does not prescribe a concrete implementation of the Object Dictionary. Only the structure of the object descriptions transmitted over the bus are defined. In addition, the existence of an Object Dictionary itself is not required if the necessary information may be constructed, e.g. with an algorithm. Thus a uniform view of communication objects is provided over the bus in a distributed application with devices of different manufacturers (multivendor facility).

A specific object description is addressed with the same logical address (index) or the same name as the corresponding communication object. Corresponding FMS services are provided to the remote application process. The object description is read with the service Get-OD. Using the Put-OD service, entries in the Object Dictionary may be registered, deleted or changed.

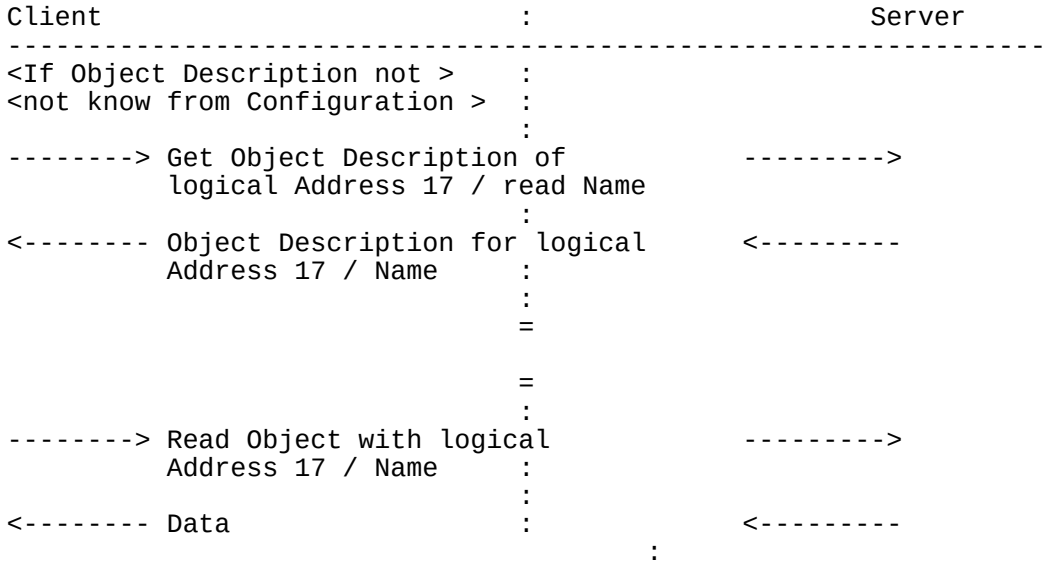


Figure 19. Example for Reading an Entry in the Object Dictionary and the Object (Variable Object)

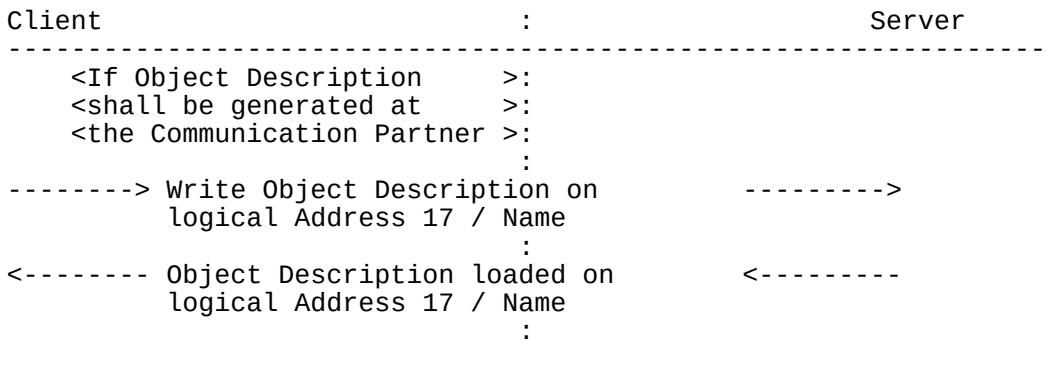


Figure 20. Example for Writing an Entry into the Object Dictionary of the Communication Partner

3.3.5.1 Description

The object description of a single communication object contains information about the object code, the assigned logical address and the name if desired. Furthermore, the assignment of the real address to the logical address and, if required, to a name are defined in the object description. The object description contains additional object specific attributes (see figure below).

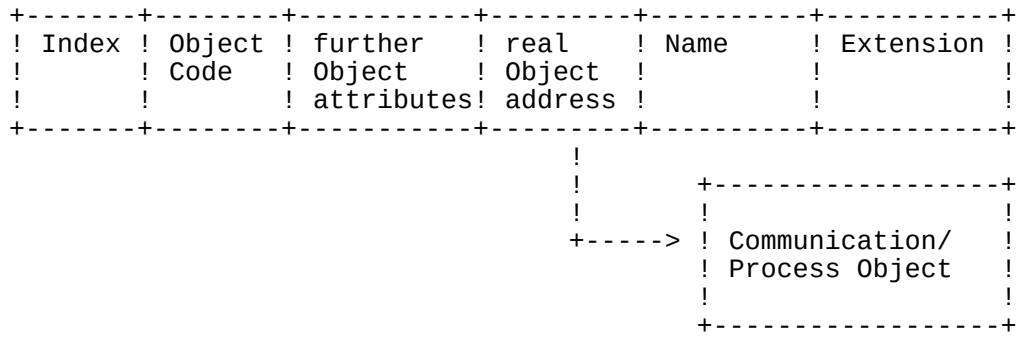


Figure 21. Example of an Object Entry as a Communication Partner sees it (Principle)

3.3.6 Protection Mechanisms

Protection mechanisms exist for communication end points and communication objects. Communication end points are protected by the communication whereas communication objects are protected by the application process.

A communication end point on connection-oriented and also on connectionless communication relationships may be protected by configuration in the Communication Relationship list. The access right to this communication end point is given only to a defined communication partner (access control).

In the case of connection-oriented communication relationships it is also possible to protect the access after successfully establishing the connection (open communication end point). Therefore several communication partners are able to use a communication end point at different times one after the other. For connection-oriented data transfer several communication partners may not simultaneously use a communication end point.

A communication object may be protected by configuration in the Object Dictionary. The access to a communication object may be allowed for certain communication partners only. Beyond that it is possible to restrict the services permissible on a communication object by an entry in the Object Dictionary (e.g. variable is read only).

Further protection mechanisms of the application process are device specific and therefore not affected by the PROFIBUS Specification.

3.4 Model of Fieldbus Management Layer 7 (FMA7)

The FMA7 describes management objects, services and the resulting models. The objects are described implicitly (by the PROFIBUS Specification). Accessing the objects is done using object specific services. One distinguishes between local and remote FMA7 services. A parallel execution of FMA7 services is not possible.

The service parameters are described analogously to the presentation in the table before.

The FMA7 is based on the FMA1/2 functions specified in PROFIBUS Data Link Layer and the management functions of LLI and FMS.

3.4.1 Local Management

Local management services are rendered locally and allow the manipulation of local management objects.

At the management user, a request is given by a request service primitive. The acknowledgement of the request is received with a confirmation service primitive. FMA7 maps local management services onto the management services of FMA1/2, LLI and FMS.

3.4.2 Remote Management

Remote management services allow the manipulation of management objects at the remote station. The FMA7 service is transmitted over the management connection using a FMA7_PDU. The remote management is a user of the LLI (like FMS). Remote management services are characterized by the remote user rendering a service by means of its local management. The remote management is connection-oriented and uses the transport functionality of the LLI.

All the PROFIBUS stations, except diagnosis and configuration tools, support remote management services as a responder on exactly one management connection only. The characteristics and the addressing of the management connection is defined at the responder. Configuration or diagnosis tools support remote management services also as a requester on several management connections.

3.4.3 Default Management Connection

With the definition of the default management connection, a uniform access to PROFIBUS stations is provided for configuration or diagnosis tools. Each PROFIBUS station that supports FMA7 remote services as a responder shall have a default management connection under CREF 1 registered in the CRL.

If a complete remote configuration of a station is required (Object Dictionary (OD), Communication Relationship List (CRL) and bus parameters), two connections have to be provided for configuration in the CRL: a management connection for the configuration of the CRL and the bus parameters, and a FMS connection for the configuration of the OD.

4 Fieldbus Message Specification (FMS)

In order to fit to a wide range of operational areas, the communication system PROFIBUS provides the application system with a variety of services for using open communication.

This variety is not required in all devices and areas. Restrictions are defined in application specific profiles. The provided services take the following criteria into consideration:

- The service scope is adapted to the operational area of the fieldbus.
- The compatibility to MAP (MMS) is guaranteed as appropriate.
- The hard time requirements shall be fulfilled.
- The service definition is with a modern, object oriented method.

4.1 Service Model

The service model of the application layer contains services that allow operations on objects.

Some of the objects are described implicitly, i.e. the description is defined by the PROFIBUS Specification. The rest are described explicitly in the Object Dictionary. Access to objects is achieved using object specific services.

When addressing objects the PROFIBUS Specification distinguishes between logical, physical and implicit addressing, and also addressing by name.

4.1.1 Short Description of Services

Initiate (establishing a connection)
This service is used to establish a connection between two communication partners. Mutual, connection specific agreements on the manner of using the communication are made (context). All other communication services are only permitted after a successful connection establishment when connection-oriented data transfer is used.

Abort (release of a connection)
With this service an existing connection between two communication partners is released. A resumption of the connection in case of connection-oriented data transfer is only possible after the connection has been established again (Initiate).

Reject (refusing an inadmissible service or PDU)
The FMS protocol uses the Reject service to refuse an inadmissible service or an inadmissible received PDU. Reasons for the Reject:

- violation of the transmission protocol
- error in the service request
- error in the response of a service
- service not executable
- service offends against the context agreement.

Status (reading the status of the device / application)
This service is used to read the status of the device / application.

UnsolicitedStatus (spontaneous reporting of the status)
This service is used for a spontaneous transmission of the device / application status.

Identify (reading of manufacturer, type and version)
This service is used to read the vendor name (name of the manufacturer), model name (name of the type of device) and revision (identification of the version) of a device.

GetOD (reading an object description)
This service is used to read a single or several object descriptions out of the Object Dictionary (OD). Thus the index or the name of the desired object description shall be given in the GetOD service or the option to read all of the object descriptions may be selected.

InitiatePutOD (opening the download of the OD)
A client uses this service to indicate to the server that it intends to deliver object descriptions.

PutOD (loading an object description)
The client uses the PutOD service to deliver object descriptions to the server. The object descriptions are loaded into the server's Object Dictionary.

TerminatePutOD (termination of the downloading of the OD)
The client uses this service to indicate to the server that the delivery of the object descriptions is complete.

InitiateDownloadSequence (opening a download sequence)
With this service a blockwise transmission of data (parameters, program code, ...) from the "client" into a domain of the "server" is announced.
The PROFIBUS Specification requires the domain to be defined in the Object Dictionary before a download can be performed. Special attributes of the domain are also stored in the OD and are not set with the InitiateDownloadSequence service.

DownloadSegment (transmitting a download data block)
The "server" uses this service to fetch a block of data from the "client". The "client" transmits the data and an additional notification as to whether further data is ready to be transmitted. The maximal block length is determined by the length of user data in a PROFIBUS frame.

TerminateDownloadSequence (terminating a download sequence)
With this service the "server" communicates that the blockwise transmission of data in its domain is complete. In addition the "server" indicates if the transmission has been successfully completed.

RequestDomainDownload (requesting a download)
With this service the "server" requests its "client" to perform a download. The request of the "server" may contain the optional information from which file of the "client" the data shall be read. The answer (acknowledgement) to that request is transmitted only after completion of the download.

InitiateUploadSequence (opening an upload sequence)
With this service the client opens the blockwise transmission of data (parameters, program code, ...) out of a domain of the server.

UploadSegment (transmitting an upload data block)
With this service the client fetches a block of data out of a domain of the server. The server transmits the data and an additional notification as to whether further data is ready to be transmitted. The maximal block length is determined by the length of user data in a PROFIBUS frame.

TerminateUploadSequence (terminating an upload sequence)
With this service the client communicates that the blockwise transmission of data from the server to the client is complete.

RequestDomainUpload (requesting an upload)
With this service the "server" requests its "client" to perform an Upload. The request of the "server" may contain the optional information as to which file of the "client" the upload data shall be written. The answer (acknowledgement) to this request is not transmitted before the Download has been completed.

CreateProgramInvocation (combine domains to a program)
With this service, domains (e.g. code, data) that are described in the Object Dictionary , are combined online to a program-invocation object. This object shall be addressed using a logical address.

DeleteProgramInvocation (deleting a program)
With this service a program invocation object is deleted.

Start (starting a program after a reset)
A program is started and executed from the beginning.

Stop (stopping a program)
A running program is stopped but not set to the beginning.

Resume (resuming a program after a stop)
A stopped program is set into the RUNNING state but not reset.

Reset (resetting a program)
A stopped program is set at the beginning.

Kill (shut down a program)
A program invocation is set into the UNRUNNABLE state independent of its present state.

Read (reading a variable)
The value of a variable object defined in the Object Dictionary is read.

Write (writing a variable)
With this service a value is assigned to a variable object described in the Object Dictionary.

ReadWithType (reading a variable)
The value and the data type description of a variable object described in the Object Dictionary are read.

WriteWithType (writing a variable)
A value is assigned to a variable object described in the Object dictionary. The data type description is appended to the value.

PhysRead (reading a physical access object)
With this service the value of a physical access object is read.

PhysWrite (writing a physical-access object)
With this service a value is assigned to the physical access object.

InformationReport (sending data - not acknowledged)
The value of a variable object described in the Object Dictionary is transmitted to e.g. all other stations for synchronization purposes.

InformationReportWithType (sending data - not acknowledged)
The value and the data type description of a variable object described in the Object Dictionary is transmitted to e.g. all other stations for synchronization purposes.

DefineVariableList (combine variables to a list)
With this service variable objects described in the static Object Dictionary are combined online to the new variable object variable list. This object is addressed thereafter using a logical address.

DeleteVariableList (deleting of a variable list)
With this service a variable list is deleted.

EventNotification (notifying an event)
With this service a predefined event message is transmitted to another communication partner. Optionally event data may be transmitted in addition.

EventNotificationWithType (notifying an event)
With this service a predefined event message is transmitted to another communication partner. Optionally event data may be transmitted in addition; in this case a data type description shall be appended to the event data.

AcknowledgeEventNotification (acknowledging an event)
With this service the receipt of an event is confirmed.

AlterEventConditionMonitoring (enable / disable event)
With this service the transmission of an event is released or suppressed.

4.1.1.1 Marginal Conditions for the Services

The following services shall be provided in all PROFIBUS devices as a server:

- Initiate
- Abort
- Reject
- Status
- Identify
- GetOD (short form only, see OD Services)

The variety of services on a communication relationship may be restricted (see LLI, section communication relationships):

- master / master or master / slave
- connection-oriented (acyclic, cyclic, each with or without slave initiative)
- connectionless (broadcast, multicast).

The variety of services may be subjected to further restrictions through application specific definitions (profiles).

Only one object may be addressed with a service. This object may in turn consist of several objects.

With one service, about 220 octets of user data at most may be transmitted.

In addition to the service specific parameters defined for each service, the communication reference shall be transferred at the interface FMS - user. The FMS delivers the communication reference to the LLI.

In the case of confirmed services, the identification of the request (Invoke-ID) shall be transferred too. An exception is the Initiate-Service where no Invoke-ID is required.

4.1.1.2 Client and Server

A client in terms of communication is an application process that, with regard to a service, uses the functionality of a VFD of a remote application process. The server is the application process that, relative to the service, provides the functionality of its VFD to the client. An application process may be server and client at the same time.

Services are provided to issue requests. Requests are transmitted to the communication partner using messages (PDUs) over a defined communication relationship.

4.1.1.2.1 Confirmed and Unconfirmed Services

There are confirmed and unconfirmed services. In case of confirmed services the client gives a request and the server confirms the execution of the request with an acknowledgement (confirmation).

Unconfirmed services are initiated by the server. In case of confirmed services, the requests issued on a communication relationship shall receive an identification (Invoke-ID) from the application process. This request identification allows a unique assignment between the given request and the corresponding acknowledgement.

The execution of the services is described in detail with service primitives and service sequences.

4.1.1.2.2 Service Primitives and Service Sequences

Confirmed Services

The following diagram shows the principle sequence between a client and a server for confirmed services:

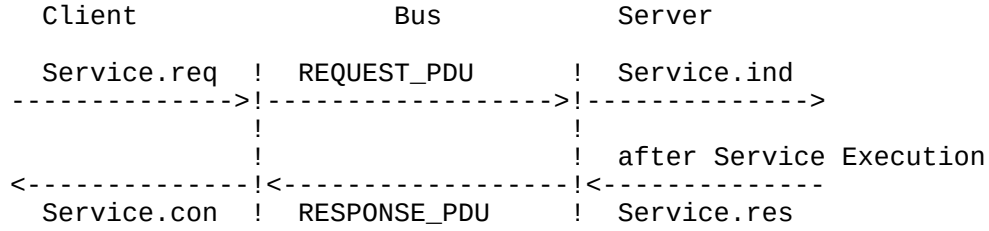


Figure 22. Sequence of Confirmed Services

A request (confirmed service) at the client is described with the service primitive request (.req) and the receipt of the server's acknowledgement of the request is described with the service primitive confirmation (.con).

At the server, the receipt of the request is described with the service primitive indication (.ind) and the delivery of the acknowledgement of the request is described with the service primitive response (.res).

Unconfirmed Services

The following diagram shows the principle sequence between a client and a server for unconfirmed services:

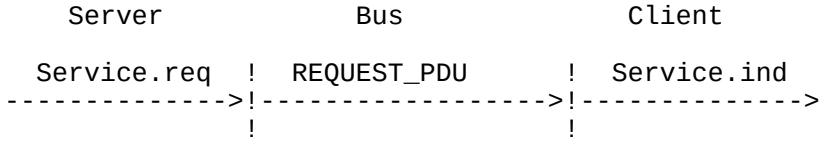


Figure 23. Sequence of Unconfirmed Services

An unconfirmed service is initiated by the server using the service primitive request (.req).

The receipt at the client is notified with the service primitive indication (.ind).

4.1.2 Short Description of Objects

Explicit Objects

The PROFIBUS Specification recognizes the following objects described explicitly by the Object Dictionary:

- Object Dictionary
- domain
- program invocation
- simple variable
- array
- record
- variable list
- event

Implicit Objects

The PROFIBUS Specification recognizes the following objects described implicitly:

- transaction-object
- VFD
- object description Object Dictionary
- data type
- data type structure description
- object description domain
- object description program invocation
- object description simple variable
- object description array
- object description record
- object description variable list
- object description event
- physical access object

The objects are described in each subclause in the following form:

```
Object:  designation of the object
  Key Attribute:  key attribute for this object
  Attribute:  attribute of the object
  Attribute:  attribute of the object
```

Attribute

The attribute expresses a certain defined characteristic of the object.

Key Attribute
The key attribute is a characteristic of the object that makes unique addressing possible.

If an object is addressed using a combination of two key attributes, then these two key attributes are given on a line separated by the character "&".

If an object is addressed alternatively by several key attributes, then these are declared on successive lines. An object may always be addressed with the first key attribute; addressing with further key attributes is optional.

4.1.2.1 Access Rights

Objects may be protected from unauthorized access. The services for accessing the object are restricted to the authorized communication partners.

Objects may only be accessed if the authorization to access the object exists. The relevant information for access control is located in the object description of the OD and in the object description of the specific objects, and is in part transmitted with the Initiate service when establishing a connection.

Access protection for the object description itself is not provided.

```
OD object description: attribute  access protection supported

object description:    attribute  password
                       attribute  access groups
                       attribute  access rights

Initiate service:      parameter access protection supported (calling)
                       parameter password (calling)
                       parameter access groups (calling)
                       parameter access protection supported (called)
                       parameter password (called)
                       parameter access groups (called)
```

Access Protection Supported
Indicates if access rights are checked when accessing an object.

access protection supported: false
Every communication partner is authorized to access every object. The attributes password, access groups and access rights do not exist in the object descriptions.

access protection supported: true
The authorization depends on the parameters transmitted with the Initiate service (password and access groups) as well as the attributes of the objects themselves (password, access groups and access rights). The object attribute access rights differentiates rights, e.g. reading, writing and executing.

Authorization is checked for every access.

- If the object attribute access rights allows access from all communication partners, the authorization exists.
- If the object attribute access rights allows access from access groups and if the client is a member of at least one of the authorized access groups, then authorization exists.
- If the object attribute access rights allows access with a password and if the password of the object description is identical to the password transmitted with the Initiate service, then authorization exists.

Password

The PROFIBUS Specification distinguishes between the attribute password in the object description and the parameter password of the Initiate service.

- attribute password (object description)
  This attribute contains an identification defined by the user that authorizes access to the object in accordance with the attribute access rights.

- parameter password (Initiate service)
  This parameter indicates the password valid for a specific communication relationship. If access with password shall not be provided on a communication relationship the parameter password shall have the value 0.
  In the case of a station supporting access with password, the values of the parameter password received with the Initiate service shall be unique for all communication relationships (exception password = 0).

Access Groups

The PROFIBUS Specification distinguishes between the attribute access groups in the object description and the parameter access groups of the Initiate service. There are 8 different access groups possible.

- attribute access groups (object description)
  This attribute contains the groups defined by the user, which authorize access to the object in accordance with the attribute access rights.

- parameter access groups (Initiate service)
  This parameter defines the membership to one or several groups which is valid for a communication relationship. If access with access groups is not to be conducted on a communication relationship, the parameter access groups shall have the binary value 00000000.

Access Rights

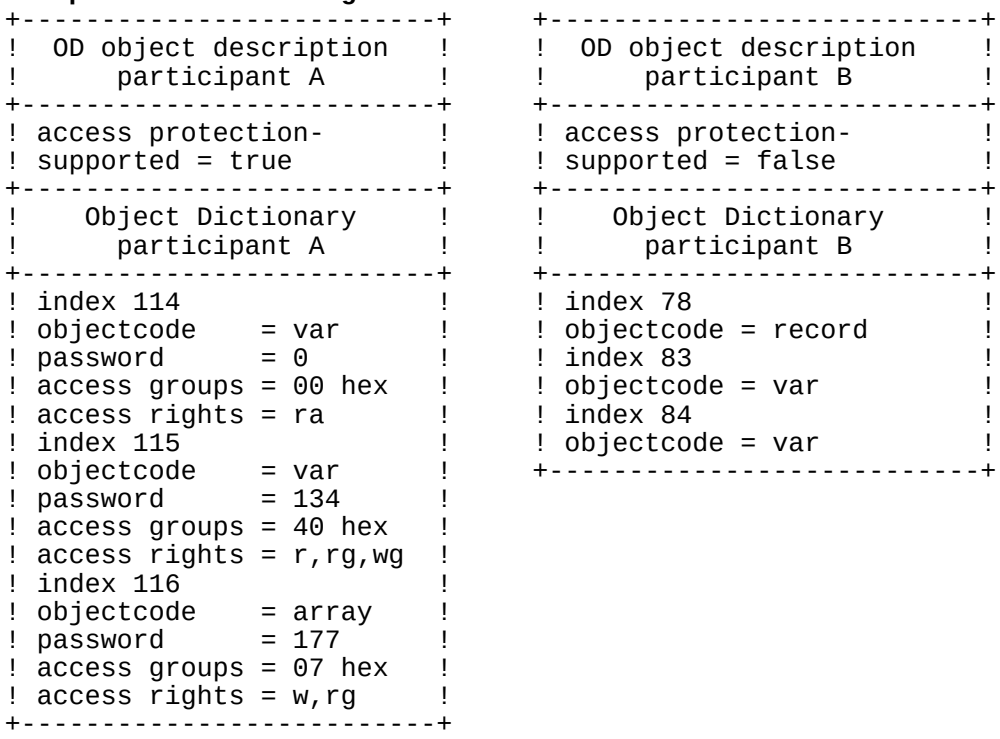
The PROFIBUS Specification differentiates rights, e.g. for reading, writing and executing, according to access with password, access for access groups and access for all communication partners. Thus it is possible to give differentiated access rights, e.g., right to read for all communication partners, right to write for access groups, right to delete for a single communication partner (password).

The following communication objects may be assigned access rights:

Table 2. Access Rights to Objects

Object	Rights
domain	upload , download , create PI
program invocation	start , stop , delete
simple variable	read , write
array	read , write
record	read , write
variable list	read , write , delete
event	alter , acknowledge

Example for Access Rights



explanation:
var simple variable
r right to read with password
rg right to read for access groups
ra right to read for all communication partners
w right to write with password
wg right to write for access groups

Initiate service, participant A as a client

```
-->  Initiate.req   access protection-supported (calling) = true
                    password (calling)          = 0
                    access groups (calling)     = 00 hex
<--  Initiate.res   access protection-supported (called) = false
                    password (called)           = 134
                    access groups (called)      = 05 hex
```

Other services, participant A as a client

```
-->  Read.req       index = 78
<--  Read.res(+)    ( access to all objects possible )
```

Other services, participant B as client

```
<--  Read.req       index = 114
-->  Read.res(+)    (all may read object 114 )
<--  Write.req      index = 114
-->  Write.res(-)   (no station may write to object 114 )
<--  Read.req       index = 115
-->  Read.res(+)    (password 134 may read )
<--  Write.req      index = 115
-->  Write.res(-)   (station B does not belong to access group 40 hex )
<--  Read.req       index = 116
-->  Read.res(+)    (access group 04 hex and 01 hex may read)
<--  Write.req      index = 116
-->  Write.res(-)   (password not correct, and access groups are
                    not authorized to write)
```

explanation of the symbols:
--> messages from station A to station B
<-- messages from station B to station A

4.1.2.2 Domain Restrictions of Objects

The domain of validity of a communication object is the domain, in which a unique addressing of the object is possible.

4.1.2.3 Creation and Deletion of Objects

All objects are created during configuration. In addition, the objects variable list and program invocation may be created or deleted dynamically using the corresponding services.

4.1.3 Addressing of Objects

Each object is addressed with a defined address scheme. The following address schemes are permissible:

- logical addressing (index)
- physical addressing
- implicit addressing
- addressing by name

4.1.3.1 Logical Addressing

The addressing of the following objects is achieved with a logical address (index):

- object description object dictionary
- object description domain
- object description program invocation
- object description simple variable
- object description array
- object description record
- object description variable list
- object description event
- domain
- program invocation
- data type
- data type structure description
- simple variable
- array
- record
- variable list
- event

The logical address is represented by an index of data type unsigned16. The relationship between index and object is stored in the object description in the Object Dictionary.

4.1.3.2 Physical Addressing

The physical access object may only be addressed using physical addressing. Physical addressing is done with an address of 32 bit, the meaning of which is defined by the addressed system.

4.1.3.3 Implicit Addressing

The object VFD is addressed by communication references (CREF). The object Object Dictionary is addressed via the VFD. The transaction object is addressed specifically to the connection using the invoke-ID and the communication reference of confirmed services.

4.1.3.4 Named Address

Addressing by name is optional. The following objects may be addressed by name in addition to logical addressing:

- object description domain
- object description program invocation
- object description simple variable
- object description array
- object description record
- object description variable list
- object description event
- domain
- program invocation
- simple variable
- array
- record
- variable list
- event

The names are represented with a visible string. The length of the visible string is fixed and it is defined in the OD object description by the attribute name length. The value of the attribute name length shall be a maximum of 32.

If no name is to be defined for an object, then the attribute name shall be set to the visible string of the defined length consisting only of SPACE characters.

Addressing by name is only possible if names are permissible in the Object Dictionary and if the field FMS Features Supported in the FMS CRL entry of the associated communication relationship indicates that the option "addressing by name" is supported.

The structure and the usage of the names is defined by the application as well as by profiles.

A name with a visible string that consists only of SPACE characters shall not be used with FMS services for addressing objects.

The names shall be unique at least within the following groups of objects:

- domain
- program invocation
- simple variable, array, record
- variable list
- event

Addressing an object by name requires that the parameter "name" given in the service matches the object's name within the associated object group in the OD. The names match if the length (indicated in the OD object description) is the same and if each character matches.

4.1.4 Assignment of Services to Master / Slave and to Objects

The following table gives an overview as to which services shall be provided by a passive station (slave), which shall be provided by an active station (master) and which services are optional. In addition the functionality of server and client is given separately. Moreover, the table distinguishes between confirmed

and unconfirmed services.

Table 3. Assignment of Services to Master and Slave

Service	Slave Cli	Slave Serv	Master Cli	Master Serv	cfd/ unc
Initiate	-	M	O	M	cfd
Abort	M	M	M	M	unc
Reject	M	M	M	M	unc
Status	-	M	O	M	cfd
UnsolicitedStatus	O	O	O	O	unc
Identify	-	M	O	M	cfd
GetOD (short form)	-	M	O	M	cfd
GetOD (long form)	-	O	O	O	cfd
InitiatePutOD	-	O	O	O	cfd
PutOD	-	O	O	O	cfd
TerminatePutOD	-	O	O	O	cfd
InitiateDownloadSequence	-	-	O	O	cfd
DownloadSegment	-	-	O	O	cfd
TerminateDownloadSequence	-	-	O	O	cfd
RequestdomainDownload	-	-	O	O	cfd
InitiateUploadSequence	-	O	O	O	cfd
UploadSegment	-	O	O	O	cfd
TerminateUploadSequence	-	O	O	O	cfd
RequestdomainUpload	-	-	O	O	cfd
CreateProgramInvocation	-	O	O	O	cfd
DeleteProgramInvocation	-	O	O	O	cfd
Start	-	O	O	O	cfd
Stop	-	O	O	O	cfd
Resume	-	O	O	O	cfd
Reset	-	O	O	O	cfd
Kill	-	O	O	O	cfd
Read	-	O	O	O	cfd
Write	-	O	O	O	cfd
ReadWithType	-	O	O	O	cfd
WriteWithType	-	O	O	O	cfd
PhysRead	-	O	O	O	cfd
PhysWrite	-	O	O	O	cfd
InformationReport	O	O	O	O	unc
InformationReportWithType	O	O	O	O	unc
DefineVariableList	-	O	O	O	cfd
DeleteVariableList	-	O	O	O	cfd
EventNotification	O	O	O	O	unc
EventNotificationWithType	O	O	O	O	unc
AcknowledgeEventNotification	-	O	O	O	cfd
AlterEventConditionMonitoring	-	O	O	O	cfd

Used abbreviations:

Cli	Client	cfd	confirmed service
Serv	Server	unc	unconfirmed service
M	Mandatory	O	Optional
-	service not allowed		

The table below lists the objects on which the specific services act.

Table 4. Assignment Services / Objects

Services	Objects										
Initiate											
Abort			T								
Reject											
Status	F		T								
UnsolicitedStatus	F										
Identify	F		T								
GetOD		O	H	T					Dt	Ds	
InitiatePutOD		O		T							
PutOD		O	H	T	D	P		V	VL	Dt	Ds	E
TerminatePutOD		O		T							
InitiateDownloadSequence		O	H	T	D						
DownloadSegment		O	H	T	D						
TerminateDownloadSequence		O	H	T	D						
RequestdomainDownload		O	H	T	D						
InitiateUploadSequence		O	H	T	D						
UploadSegment		O	H	T	D						
TerminateUploadSequence		O	H	T	D						
RequestdomainUpload		O	H	T	D						
CreateProgramInvocation		O	H	T	D	P					
DeleteProgramInvocation		O	H	T	D	P					
Start				T		P					
Stop				T		P					
Resume				T		P					
Reset				T		P					
Kill				T		P					
Read				T				V	VL		
Write				T				V	VL		
ReadWithType				T				V	VL		
WriteWithType				T				V	VL		
PhysRead				T			Y				
PhysWrite				T			Y				
InformationReport								V	VL		
InformationReportWithType								V	VL		
DefineVariableList		O	H	T					VL		
DeleteVariableList		O	H	T					VL		
EventNotification											E
EventNotificationWithType											E
AcknowledgeEventNotification				T							E
AlterEventConditionMonitoring		O		T							E

Abbreviations:

F	VFD	Y	phYsical access object
O	Object Dictionary (OD)	V	simple Variable, array, record
H	OD object description	VL	Variable List
T	Transaction object	Dt	Data type
D	domain	Ds	Data type structure description
P	Program invocation	E	Event

4.1.5 Data Types

The following data types are standardized in the PROFIBUS Specification as communication objects:

- boolean
- integer
- unsigned
- floating point
- visible string
- octet string
- bit string
- date
- time of day
- time difference

In addition, PROFIBUS permits configured data types.

Physically the types consist of one or several octets (bytes). An octet consists of 8 bits, numbered from 1 to 8. Bit 1 is the LSB (Least Significant Bit).

For the coding of the different data types see the coding in part 6.

4.2 VFD Support

4.2.1 Model Description

The Virtual Field Device (VFD) is an abstract model for the description of the data and the behaviour of an automation system as seen by a communication partner.

The basis of the VFD Model is the VFD Object. The VFD Object contains all objects and object descriptions which may be used by a communication user via services.

The object descriptions reside in the Object Dictionary (OD). For each VFD there is exactly one Object Dictionary.

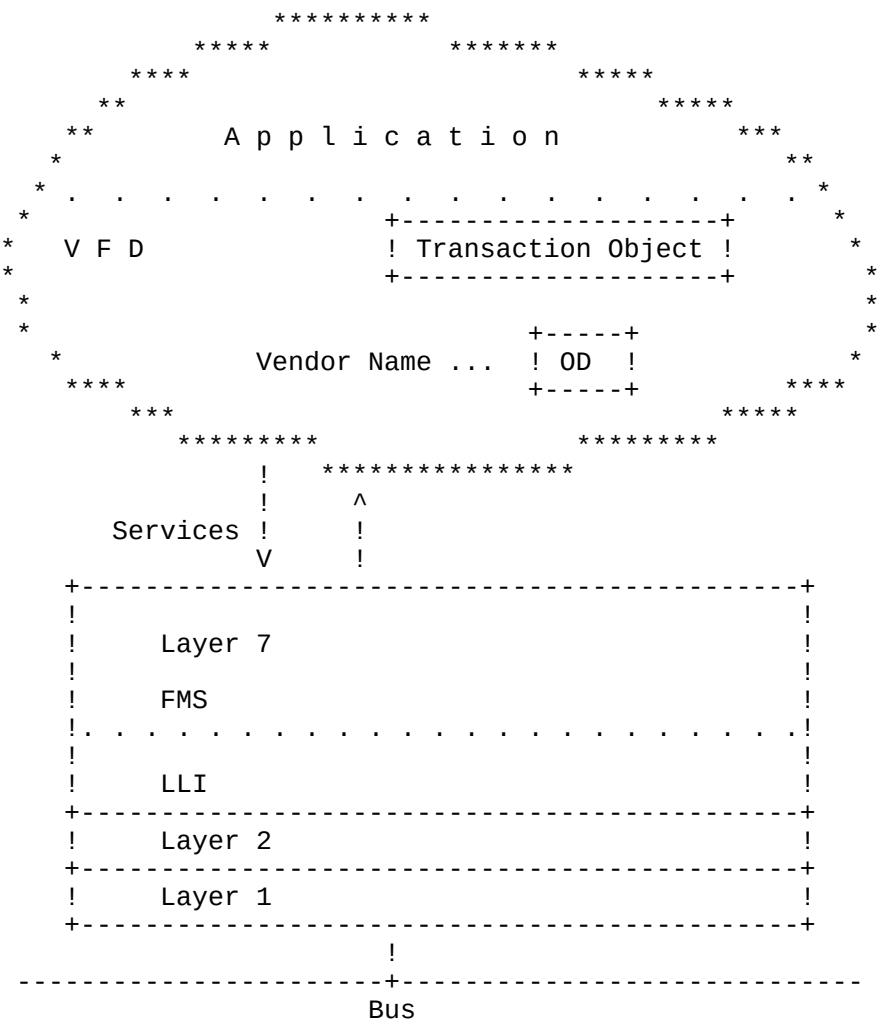


Figure 24. VFD Model

The services of Layer 7 define no concrete interface for an implementation. They describe in an abstract form which functions may be used.

The application is not a subject of this specification. It shall only be indicated, how the abstractly described services may be made available to the application.

The addressing of the VFD objects is defined implicitly by the Communication Relationships. As a rule only one VFD Object is used for each device. An extension

to several VFD Objects for each device is possible.

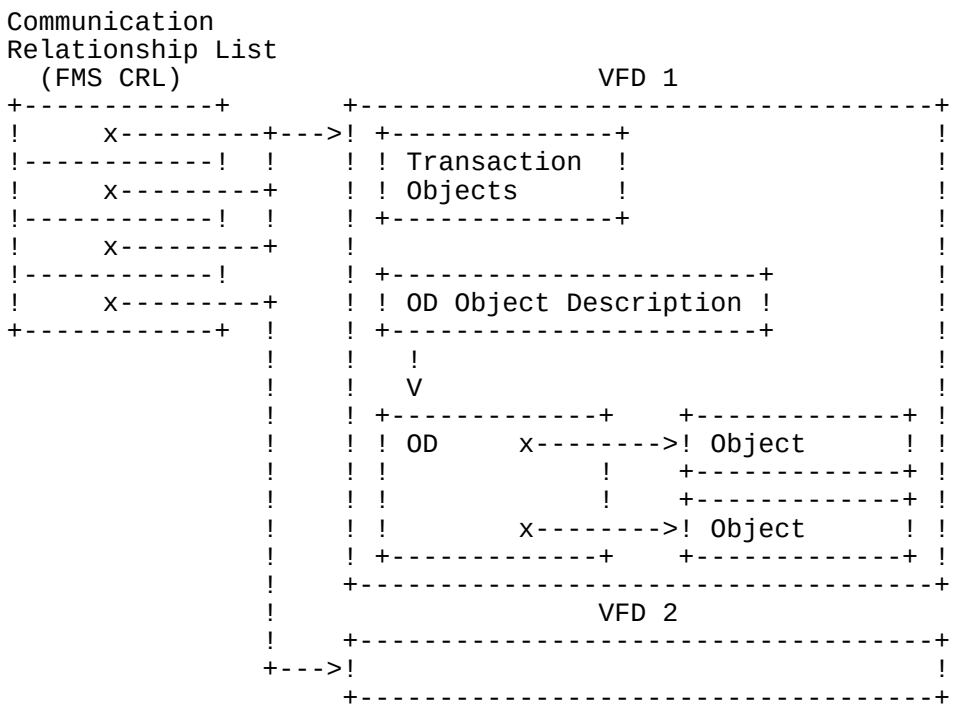


Figure 25. Abstract Model of an Automation System (VFD)

The following VFD Support Services are defined:

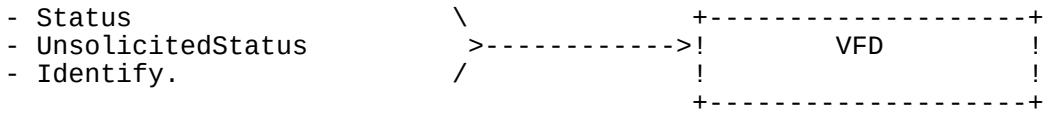


Figure 26. VFD Support Services

Specific state information of the device may be given to the application with the Status / UnsolicitedStatus. With the Identify Service the appropriate VFD can be identified for this Communication Relationship.

4.2.2 The VFD Object

The VFD Object is implicitly defined. The object description of the VFD may not be remotely read. The VFD Object is identified by the attributes

- VendorName
- ModelName
- Revision
- ProfileNumber

If there are several VFDs, it is clear from the entry in the FMS CRL, which VFD shall be addressed.

4.2.2.1 Attributes

Object: VFD
Key Attribute: implicit through the Communication Relationship
Attribute: VendorName
Attribute: ModelName
Attribute: Revision
Attribute: ProfileNumber
Attribute: LogicalStatus
Attribute: PhysicalStatus
Attribute: List of VFD Specific Objects

Implicit through the Communication Relationship:
The Communication Relationship which is entered in the FMS Communication Relationship List (see clause 3.4.2), points to the assigned VFD Object.

VendorName:
This attribute, of type visible string, identifies the vendor of the device.

ModelName:
In this visible string the model name of the device is entered by the vendor.

Revision:
A visible string, which describes the revision level of the device. The value of this string is defined by the vendor.

ProfileNumber:
The profile identification is entered in this attribute by the vendor in an octet string with a fixed length of 2 octets. If the device does not correspond to a profile, then the value "0" shall be entered in both octets.

Logical Status:
This attribute contains information about the state of the communication functionality of the device.

0 <=> ready for communication
2 <=> limited number of services
4 <=> OD-LOADING-NON-INTERACTING
5 <=> OD-LOADING-INTERACTING

Ready for communication

- All services may be used normally.

Limited number of services
The server supports at least the following services:
- Initiate
- Abort
- Reject
- Identify
- Status
- GetOD

OD-LOADING-NON-INTERACTING

If the Object Dictionary is in the state OD-LOADING-NON-INTERACTING, it is not allowed to execute the service InitiatePutOD.

OD-LOADING-INTERACTING

If the Object Dictionary is in the state OD-LOADING-INTERACTING, then all connections, except for the connection over which the InitiatePutOD Service was received, are locked and the establishment of a further connection shall be rejected. On this connection only the following services are allowed:

- Initiate
- Abort
- Reject
- PhysWrite
- GetOD
- InitiatePutOD

- Status
- Identify
- PhysRead
- PutOD
- TerminatePutOD

Physical Status:
This attribute gives a coarse summary of the state of the real device.
0 <=> operational
1 <=> partially operational
2 <=> not operational
3 <=> needs maintenance

List of VFD Specific Objects contains all VFD Specific Objects.

4.2.3 VFD Support Services

4.2.3.1 Status

The device / user state is read with the Status service.

Table 5. Status

Parameter Name	.req	.res
	.ind	.con
Argument	M	
Result(+)		S
LogicalStatus		M
PhysicalStatus		M
LocalDetail		U
Result(-)		S
ErrorType		M

Argument:
The argument does not contain service specific parameters.

Result(+):
The parameter Result(+) indicates that the service was successfully executed.

LogicalStatus:
This parameter contains information about the state of the communication functionality of the device.

PhysicalStatus:
This parameter gives a coarse summary of the state of the real device.

LocalDetail:
This parameter gives the local state of the application and the device. The meaning of the individual bits shall be defined by profiles.

Result(-):
The parameter Result(-) indicates that the service was not executed successfully.

Error-Type:
Contains the reason for the unsuccessful execution of the service.

4.2.3.2 UnsolicitedStatus

The UnsolicitedStatus service is used by the user for a spontaneous transmission of the device / user state.

Table 6. UnsolicitedStatus

Parameter Name	.req / .ind
Argument	M
Priority	M
LogicalStatus	M
PhysicalStatus	M
LocalDetail	U

Argument:
The Argument contains the service specific parameters of the service call.

Priority:
This parameter gives the priority of this service. It shall have one of the values
true : high priority
false : low priority

LogicalStatus:
This parameter contains information about the state of the communication functionality of the device.

PhysicalStatus:
This parameter gives a coarse summary of the state of the real device.

LocalDetail:
This parameter indicates the local state of the application and the devices. The meaning of the individual bits shall be defined by profiles.

4.2.3.3 Identify

Information to identify a VFD is read with this service. The attribute ProfileNumber of the VFD shall be transmitted with the Initiate service.

Table 7. Identify

Parameter Name	.req / .ind	.res / .con
Argument	M	
Result(+)		S
VendorName		M
ModelName		M
Revision		M
Result(-)		S
ErrorType		M

Argument:
The argument contains no service specific parameters.

Result(+):
The parameter Result(+) indicates that the parameters could be read.

VendorName :
This parameter contains the vendor name in the VFD attribute.

ModelName :
This parameter gives the model name.

Revision:
This parameter contains the Revision attribute of the VFD.

Result(-):
The parameter Result(-) indicates that the service could not be executed.

ErrorType:
Contains information about the reason the service could not be executed.

4.3 Object Dictionary (OD) Management

4.3.1 Model Description

The Object Dictionary (OD) contains the Object Descriptions of the following Communication Objects:

- DataType
- DataTypeStructureDescription
- Domain
- ProgramInvocation
- SimpleVariable
- Array
- Record
- VariableList
- Event

In addition the OD may contain Null objects. Null objects are defined in the following cases:
- a Null Object is configured as a placeholder in the OD,
- a Null Object is created if an existing object in the OD has been deleted by the Put OD Service, and
- a Null object shall be configured instead of a standard data type if this standard datatype is not supported.

See also Data Type Object definition.

An object description "Object Dictionary" (OD object description) is uniquely assigned to the Object Dictionary. The OD object description contains information about the structure of the Object Dictionary. The object descriptions are marked with an unique index of type unsigned16.

A name of type visible string may also be assigned to the following objects:
- Domain
- ProgramInvocation
- SimpleVariable
- Array
- Record
- VariableList
- Event

The index or the name serves as a key to the object and the object description.

The services address on principle the object of the server, because the real object exists there.

Before a remote communication partner can access an object, he shall know its description. The logical address (Index) or the name may be used, when objects are accessed remotely.

The object descriptions may be specified during system configuration, but they may also be transferred between stations at any later time.

The definition of the object description takes place at that station, in which the object really exists (Source OD). The length of an object description may be different for the individual objects. The communication partners maintain a complete or a partial copy of the Remote Object Description (Remote OD).

Thus every station may possess a Source OD for locally existing communication objects and one or more Remote ODs. The Remote ODs are connection specific, i.e. for each connection another Remote OD may be valid.

The OD model provides corresponding services for reading and writing of Source ODs. If a Source OD is modified by a remote communication partner, then this shall be notified to all other communication partners which use the same data base (Source OD). For this purpose, the user releases the connections to all other communication partners. After the OD has been modified, the connections may be established again. During the connection establishment, the new version number (Version OD) is given to each communication partner to notify that the corresponding Source OD has been modified. The addition of object descriptions is permissible without connection release.

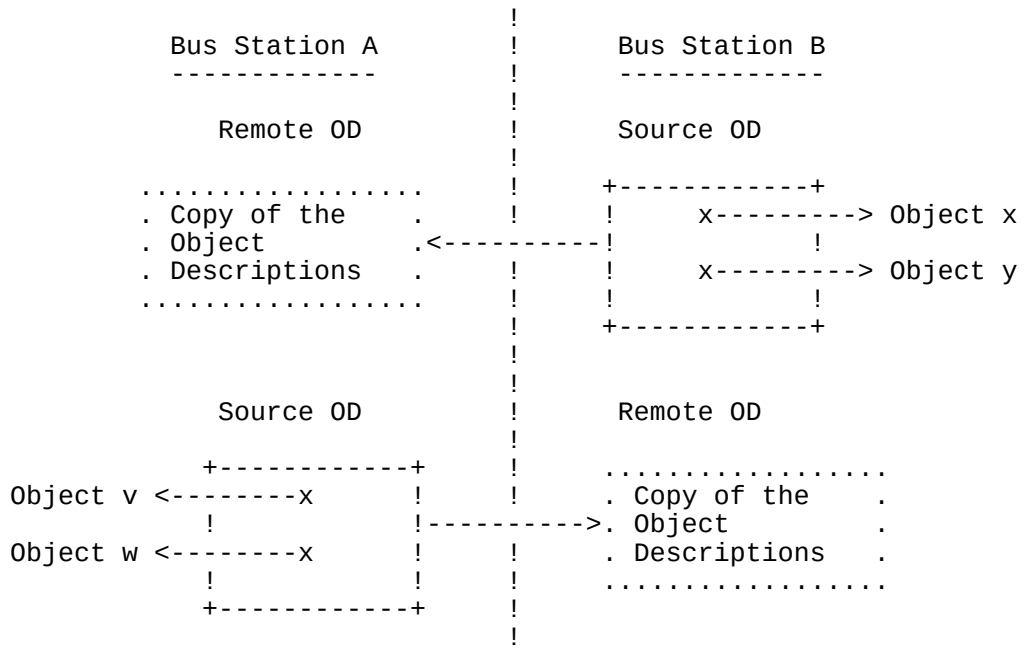


Figure 27. Source OD / Remote OD

The structure of the OD described as follows does not prescribe a concrete implementation of the OD. It only defines the structure of the object descriptions which are transmitted via the bus. Even the existence of an OD is not necessary, if the needed information can be obtained in an other way (e.g. by an algorithm).

4.3.2 Structure of an OD

The Object Dictionary (OD) is divided into several parts:
- object description "Object Dictionary" (OD object description)
  contains structural information about the OD (see OD Object Description).
- Static Type Dictionary (ST-OD)
  contains Data Types and Data Type Structure Description.
- Static Object Dictionary (S-OD)
  contains object descriptions of Simple Variables, Arrays,
  Records, Domains and Events.
- Dynamic List of Variable Lists (DV-OD)
  contains object descriptions of variable lists.
- Dynamic List of Program Invocations (DP-OD)
  contains object descriptions of Program Invocations.

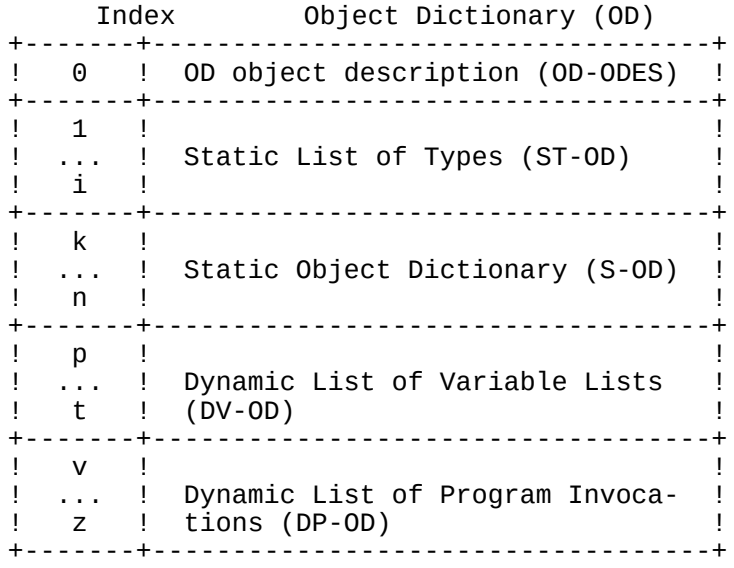


Figure 28. Structure of the Object Dictionary

4.3.2.1 Static List of Types (ST-OD)

The Static List of Types (ST-OD) contains the object descriptions of Data Types and Data Type Structure Descriptions for the Variable Access objects of the S-OD. Exactly one object description is assigned to every index. This assignment is static and may not be deleted. The ST-OD may be loaded by the remote partner (see OD Services).

The Static List of Types always begins with the index 1 in the Object Dictionary. The remaining object descriptions shall follow with increasing index. The number of assignments "index <=> object description" is entered as length of the ST-OD in the Object Dictionary.

The ST-OD is divided in two segments. The first segment contains the object descriptions of standardized PROFIBUS Data Types (Standard Data Types) and begins with index 1. The second segment follows immediately and contains object descriptions of Data Types and Data Type Structure Descriptions, which can be defined individually. This segment may be further specified by user groups with the help of profiles. The representation of the object descriptions "Data Types" and "Data Type Structure Description" in the OD may be found in the Variable Access Objects definition.

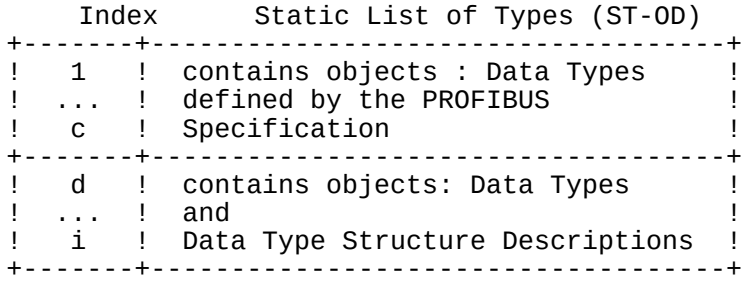


Figure 29. Structure of the Static List of Types

4.3.2.2 Static Object Dictionary (S-OD)

The Static Object Dictionary (S-OD) contains the object descriptions of Simple Variables, Arrays, Records, Domains and Events.

An object description is assigned to each index. The first index, which is associated to an object description, is entered in the object description of the Object Dictionary. The remaining object descriptions shall follow with increasing index. The number of assignments "index <=> object description" is entered as length of the S-OD in the object description of Object Dictionary.

Besides the assignment "index <=> object description", an additional assignment "name <=> object description" may exist. The length of the name may be from 0 to 32 octets. It is entered in the Name Length Field of the OD object description. If this field contains a "0", no names are present. The objects which are defined in the S-OD may be provided with access rights. Whether Access Rights are possible is entered in the field AccessProtectionSupported of the OD object description.

An object description in the S-OD contains an extension, which may contain further object specific definitions. The representation of the object descriptions in the S-OD is defined in conjunction with the domain objects, variable access objects and event object.

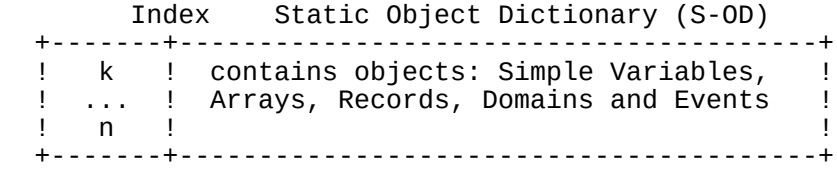


Figure 30. Structure of the Static Object Dictionary

4.3.2.3 Dynamic List of Variable Lists (DV-OD)

The Dynamic List of Variable Lists (DV-OD) contains the object description of the Variable Lists.

An object "Variable List" and its object description in the DV-OD is dynamically created with the service DefineVariableList and may be deleted with the service DeleteVariableList. Upon creation of the object access rights may be assigned to it. Moreover it is possible, to create these objects by loading the whole OD (see OD management).

The first index, which has the object description of a Variable List assigned to it, and the number of available indices are entered in the OD object description.

Fundamental information is entered in the object description about the Variable List, for example access rights, the number of Variable Access objects and the logical addresses (indices) of Variable Access objects in the S-OD.

Besides the assignment "index <=> object description", an additional assignment "name <=> object description" may exist (see static list of objects).

The representation of the object description of a Variable List in the DV-OD may be found in the definition of variable access objects.

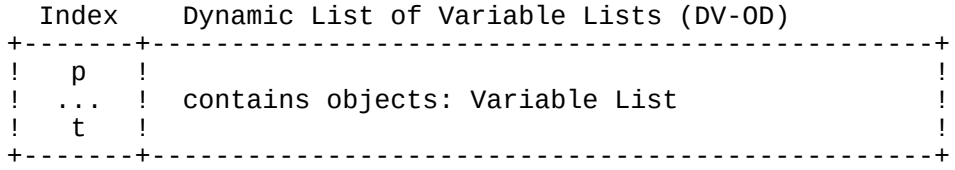


Figure 31. Structure of the Dynamic List of Variable Lists

4.3.2.4 Dynamic List of Program Invocations (DP-OD)

The Dynamic List of Program Invocations (DP-OD) contains the object descriptions of the objects Program Invocation.

An object Program Invocation and its object description in the DP-OD is dynamically created with the service Create Program Invocation and deleted with the service Delete Program Invocation. Upon creation of the object access rights may be assigned to it. Moreover it is possible to create these objects by loading the whole OD (see OD management).

The first index, which has the object description of a Program Invocation assigned to it, and the number of available indices are entered in the OD object description.

The object description of a Program Invocation contains information such as access rights and the number of Domain objects and their logical addresses (indices).

Besides the assignment "index <=> object description", an additional assignment "name <=> object description may exist (see static list of objects).

The DP-OD may contain a segment with pre-configured Program Invocations.

The representation of the object description of a Program Invocation in the DP-OD may be found in definition of the Program Invocation object.

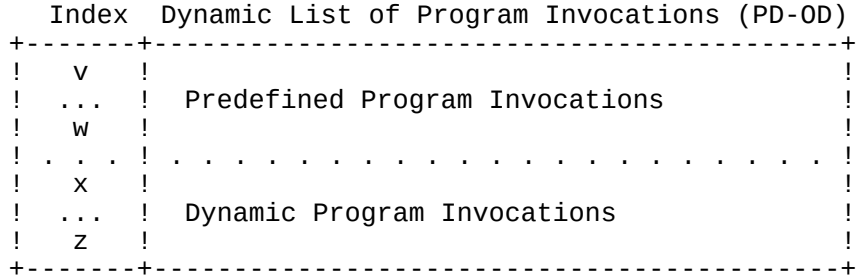


Figure 32. Structure of the Dynamic List of Program Invocations

4.3.3 Object Description in the OD

The object descriptions are entered in the OD. Every object description contains an index, an object code, further object attributes, system specific references to the real object and, if need be, a name and an extension.

The object is addressed by the index and identified by the object description. The object code is the identifier of the object and indicates which class this object belongs to. The other object attributes are object specific.

The local address serves the internal addressing of the object. Besides the index, optionally a name for the addressing of the object may also be used. The extension contains a length information for the extension and optionally further object attributes. The length information shall be transmitted as the first field. The value of the length information shall contain the length of the extension without the length field itself and it may vary from 0 to 200 octets. The value "0" for the length means: no further object attributes.

The mapping of the individual object descriptions onto the parameter "object description" (data type packed) of the services Get OD and PutOD is shown graphically in the corresponding subclauses. The representation contains two lines. The names of the object attributes are in the upper line. The lower line contains example entries. If the length of the lines is insufficient, the representation is continued on the following lines.

The attributes shall be coded corresponding to their data type (see FMS coding).

The length of the entry of one object description is variable within the maximum PDU size.

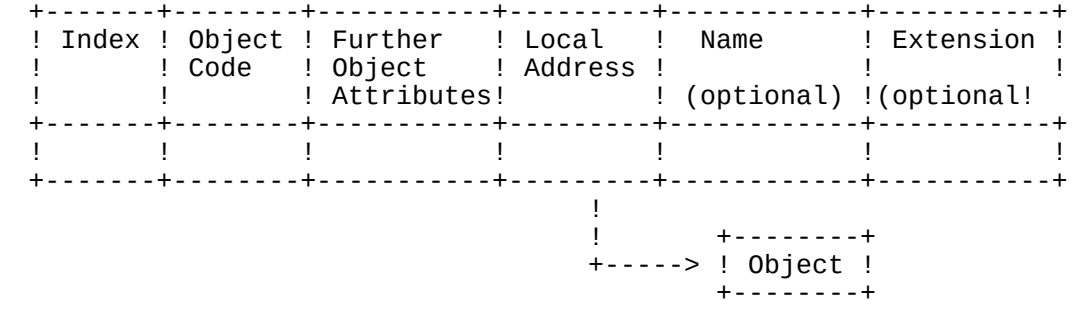


Figure 33. Structure of an Object Description in the OD

4.3.4 OD Object Description

The structure of the Object Dictionary is described by the object description "Object Dictionary" (OD object description). This has the index "0" in the Object Dictionary. By reading the OD object description the structure description and the version number of the OD are made available.

4.3.4.1 Attributes

Object: OD object description
- Key Attribute: Index
- Attribute: ROM/RAM-Flag
- Attribute: NameLength
- Attribute: AccessProtectionSupported
- Attribute: VersionOD
- Attribute: LocalAddress-OD-ODES
- Attribute: ST-OD-Length
- Attribute: LocalAddress-ST-OD
- Attribute: FirstIndex-S-OD
- Attribute: S-OD-Length
- Attribute: LocalAddress-S-OD
- Attribute: FirstIndex-DV-OD
- Attribute: DV-OD-Length
- Attribute: LocalAddress-DV-OD
- Attribute: FirstIndex-DP-OD
- Attribute: DP-OD-Length
- Attribute: LocalAddress-DP-OD

Index
Index of the Object "Object-Description Object Dictionary", here always = 0.

ROM/RAM-Flag
This attribute of type boolean describes whether modifications in the OD are allowed.
- false <=> no modifications in the OD are permitted
- true <=> modifications in the OD are permitted

NameLength
This field gives the length of the name and may take only the values "0..32":
- 0 <=> no names are used

AccessProtectionSupported
This attribute states, whether access rights for password, access groups and all communication partners are supported (true), or whether the access to all objects is permitted for every communication partner (false).

VersionOD
Gives the version of the OD.

LocalAddress-OD-ODES
This attribute is a system specific reference to the real OD object description and serves the internal addressing. If no representation of this kind is used, this attribute is set to the value FFFFFFFF hex.

ST-OD-Length

This attribute gives the maximal number of available entries in the Static List of Types. The first available index of the Static List of Types is 1, i.e. the highest available index in the Static List of Types is equal to its length.

LocalAddress-ST-OD
This attribute is a system specific reference to the real ST-OD and serves the internal addressing. If no representation of this kind takes place, this attribute is set to the value FFFFFFFF hex.

FirstIndex-S-OD
The first available index in the Static Object Dictionary.

S-OD-Length
This attribute gives the maximum number of available entries in the Static Object Dictionary. The highest available index shall be StartIndex + Length - 1.

LocalAddress-S-OD
This attribute is a system specific reference to the real S-OD and serves the internal addressing. If no representation of this kind takes place, this attribute is set to the value FFFFFFFF hex.

FirstIndex-DV-OD
First available index in the Dynamic List of Variable Lists.
0 <=> no Dynamic List of Variable Lists.

DV-OD-Length
This attribute gives the maximum number of available entries in the Dynamic List of Variable Lists. The highest available index shall be StartIndex + Length - 1.

LocalAddress-DV-OD
This attribute is a system specific reference to the real DV-OD and serves the internal addressing. If no representation of this kind takes place, this attribute is set to the value FFFFFFFF hex.

FirstIndex-DP-OD
First available index in the Dynamic List of Program Invocations.
0 <=> no Dynamic List of Program Invocations.

DP-OD-Length
This attribute gives the max. number of available entries in the Dynamic List of Program Invocations. The highest available index shall be StartIndex+Length-1.

LocalAddress-DP-OD
This attribute is a system specific reference to the real DP-OD and serves the internal addressing. If no representation of this kind takes place, this attribute is set to the value FFFFFFFF hex.

4.3.4.2 Representation of the OD on Transmission

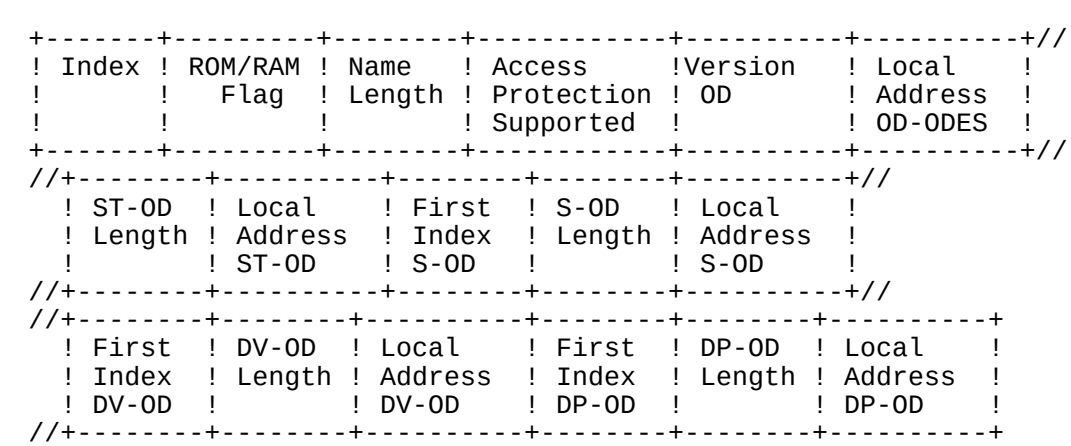


Figure 34. Structure of the OD Object Description

4.3.4.3 Empty Object Dictionary

The Object Dictionary is loadable. Therefore if an Object Dictionary is not yet loaded, at least an empty Object Dictionary shall exist.

An empty Object Dictionary is defined by pre-setting the following attributes of the OD object description and the ST-OD. For all standard data types in the ST-OD a Null object shall be configured.

Table 8. OD object description of an Empty Object Dictionary

Attribute	Value
ROM/RAM-Flag	True
NameLength	0
AccessProtectionSupported	False
VersionOD	0
ST-OD-Length	14
FirstIndex-S-OD	15
S-OD-Length	0
FirstIndex-DV-OD	0
DV-OD-Length	0
FirstIndex-DP-OD	0
DP-OD-Length	0

4.3.5 The OD Object

4.3.5.1 Attributes

Object: OD
  Key Attribute: implicit by VFD
  Attribute: OD object description
  Attribute: List of object description

implicit by VFD
The communication relationship, which is entered in the FMS Communication Relationship List (see context management), points to the assigned Object Dictionary.

OD object description
Contains the object description of the Object Dictionary.

List of object description
Contains object descriptions of the following objects:

- Domain
- ProgramInvocation
- SimpleVariable
- Array
- Record
- VariableList
- DataType
- DataTypeStructureDescription
- Event
- Null

4.3.6 OD Services

The OD services act on the object descriptions of the objects Domain, Program Invocation, Simple Variable, Array, Record, Variable List, Event, Object Dictionary, Data Type and Data Type Structure Description.

The following OD services are defined:

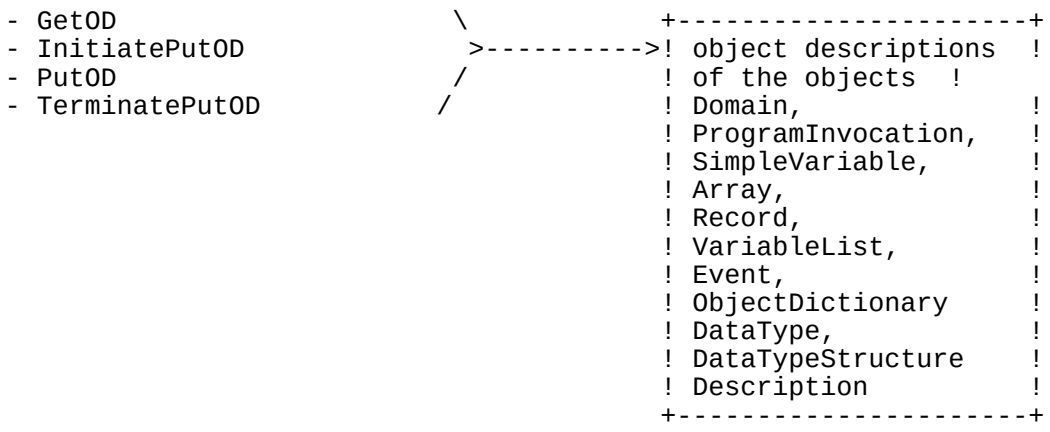


Figure 35. OD Services

With the GetOD service one or several object descriptions are read. With the PutOD service one or several object descriptions are written. The number of object descriptions, which are

transferable with one PutOD or GetOD Service, depends on their length and the maximum available PDU size.

The write access to the OD shall be initiated with an InitiatePutOD service and terminated with a TerminatePutOD service.

4.3.6.1 GetOD

With the GetOD service one or several object descriptions are read. The service distinguishes between a short and a long form. The long form of the GetOD service is optional. To read a single object description, the index or the name shall be given.

To read several or all object descriptions, the index of the first required object description (StartIndex) shall be given.

To read the whole OD, the GetOD service shall be used repeatedly.

Table 9. GetOD

Parameter Name	.req .ind	.res .con
Argument	M	
AllAttributes	M	
AccessSpecification	M	
Index	S	
VariableName	S	
VariableListName	S	
DomainName	S	
PI-Name	S	
EventName	S	
StartIndex	S	
Result(+)		S
List of Object Description		M
MoreFollows		M
Result(-)		S
ErrorType		M

Argument
The argument contains the service specific parameters of the service call.

AllAttributes
This parameter states whether the object description shall be transferred in the short form (false) or in the long from (true).

The following object attributes are not contained in the short form:

- Description
- Password
- AccessGroups
- AccessRights
- LocalAddress
- Name
- LocalAddress-OD-ODES
- LocalAddress-ST-OD
- LocalAddress-S-OD
- LocalAddress-DV-OD
- LocalAddress-DP-OD
- Extension

AccessSpecification
This parameter states which object description is accessed.

Index
Index of the object description.

VariableName
Name of a variable.

VariableListName
Name of a Variable List.

PI-Name
Name of a Program Invocation.

DomainName
Name of a Domain.

EventName
Name of an Event.

StartIndex
Index, starting with which the object descriptions shall be read.

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

List of object description:
For logical access and for access by name one, otherwise several complete object descriptions. The number depends on the maximum possible PDU size and the length of the single object descriptions. If access rights are not supported (Access Protection = false), the attributes Password, Access Groups and Access Right are meaningless.
If a selected index in the OD represents a Null object, the appropriate code for the Null object shall be returned.
If the start index is used, then the code for the Null object shall only be returned in the cases that the Null object represents a not supported standard data type. In all other cases the Null objects may be skipped.

MoreFollows
States whether further object descriptions exist after reading several object descriptions (StartIndex). For the access to single object descriptions this parameter is always = false.

Result(-)
The Result(-) parameter indicates that the service was not executed successfully.

ErrorType:
This parameter contains information about the reason for the unsuccessful execution of the service.

4.3.6.2 PutOD Services

With the PutOD services one or several object descriptions are written. Before the beginning of a modification of the OD an user may take appropriate measures (e.g. to put a process into a safe state). The communication does not test the plausibility of the new Object Dictionary.

Writing in the OD is performed with the sequence InitiatePutOD service, one or several PutOD services and a TerminatePutOD service. For this it is distinguished between a loading which is free of interactions and a loading which is not free of interactions (OD-LOADING-NON-INTERACTING and OD-LOADING-INTERACTING). The loading which is not free of interactions differentiates between a new loading (complete) and a post loading (partial). If need be, new communication objects and their state machines are created with the TerminatePutOD.

Loading free of interactions (OD-LOADING-NON-INTERACTING):

The client communicates to the server with the parameter Consequence = 0, whether the intended modifications or extensions are free of interactions for the other communication partners. In this case the communication relationships

are continued.

EXAMPLE: Modifications free of interactions are:

- Addition of new object descriptions in the static segment
- Deletion of private object descriptions for stations which are no longer in the communication system

The communication does not check whether the write to the OD is really free of interactions. The loading free of interaction is initiated with the service InitiatePutOD with the parameter Consequence = 0. After that object descriptions may be loaded with PutOD. As long as the loading process is not terminated with TerminatePutOD, every further InitiatePutOD is rejected.

Loading not free of interactions (OD-LOADING-INTERACTING):

The loading not free of interactions is initiated with the service InitiatePutOD with the Parameter Consequence = 1 (post-loading) or Consequence = 2 (new-loading). In this case the user shall release all other communication relationships (abort with Reason Code 6 shall be issued). After that the object description may be loaded with PutOD. It is also the task of the client user to enter the new version number in the OD. The loading process is terminated with the service TerminatePutOD. After that all communication relationships may be re-established. With the re-establishment of the communication relationships the new version number of the OD is transferred. The communication partners may then recognize that the corresponding OD has been modified. Whereas for post-loading solely parts of the OD are loaded, the existing OD is deleted and subsequently loaded completely anew upon receipt of an InitiatePutOD with Consequence = 2.

4.3.6.2.1 InitiatePutOD

The InitiatePutOD opens the process of loading the OD free of interaction or not free of interaction.

Table 10. InitiatePutOD

Parameter Name	.req / .ind	.res / .con
Argument	M	
Consequence	M	
Result(+)		S
Result(-)		S
ErrorType		M

Argument
The argument contains the service specific parameters of the service call.

Consequence
This parameter states whether the intended modifications are free of interaction for the other communication partners (Consequence = 0), not free of interaction with partial reload of the OD (Consequence = 1) or not free of interaction with completely new load of the OD (Consequence = 2).

0 <=> loading free of interaction
1 <=> reload, not free of interaction
2 <=> newload, not free of interaction

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

Result(-)
The Result(-) parameter indicates that the service was not executed successfully.

ErrorType
This parameter contains information about the reason for the unsuccessful execution of the service.

4.3.6.2.2 PutOD

With the PutOD service one or several object descriptions are written into the OD.

Table 11. PutOD

Parameter Name	.req	.res
	.ind	.con
Argument	M	
List of Object Description	M	
Result(+)		S
Result(-)		S
ErrorType		M

Argument
The argument contains the service specific parameters of the service call.

List of object description
One or several object descriptions which shall be entered into the OD. If access rights are not supported (Access Protection = false), the attributes Password, Access Groups and Access Rights are meaningless.
An object description may also be deleted with the PutOD service. In this case, an empty object description with the following structure is transferred:
- Index of the object description to be deleted
- Object Code = Null Object.

Index	Object Code
246	Null Object

Figure 36. Empty Object Description (Example)

Result(+)
The Result(+) parameter indicates that the service was successfully executed.

Result(-)
The Result(-) parameter indicates that the service was not successfully executed. In this case, no object description is written into the OD.

ErrorType
This parameter contains information about the reason for the unsuccessful execution of the service.

4.3.6.2.3 TerminatePutOD

The TerminatePutOD service terminates the process of loading of the OD. The objects with their state machines are generated.

Table 12. TerminatePutOD

Parameter Name	.req	.res	.ind	.con
Argument	M			
Result(+)		S		
Result(-)		S		
ErrorType		M		
Index		C		

Argument
The Argument contains no service specific parameters.

Result(+)
The Result(+) parameter indicates that the service was successfully executed.

Result(-)
The Result(-) parameter indicates that the service was not successfully executed. If an error occurs during the generation of the objects, the old state is restored.

ErrorType
This parameter contains information about the reason for the unsuccessful execution of the service.

Index
This parameter gives the index of the object, for which the generation was not successful.

4.3.7 State Machine

4.3.7.1 State Machine Description

OD-LOADING-NON-INTERACTING

In this state the execution of the service InitiatePutOD is not permitted. In this state the service PutOD on the DV-OD or the DP-OD is not permitted.

OD-LOADING-INTERACTING

Except for the connection over which the InitiatePutOD service was received, all connections shall be aborted and the establishment of a further connection shall be rejected with an abort using Reason Code 6. On this connection only the following services are allowed:

- Initiate
- Abort
- Reject
- PhysWrite
- GetOD
- InitiatePutOD

- Status
- Identify
- PhysRead
- PutOD
- TerminatePutOD

OD-READY

In this state all connections and services may be used normally.

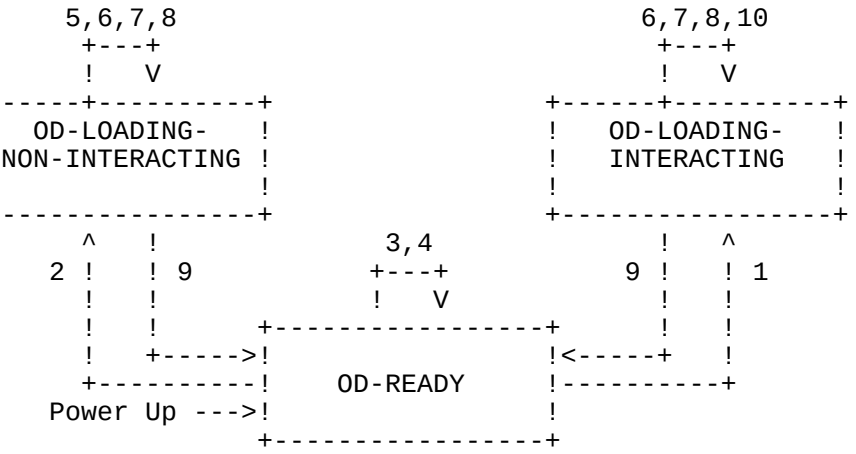


Figure 37. State Machine

4.3.7.2 State Transitions

	Event / Exit Condition => Action Taken
1	InitiatePutOD.ind \consequence >0 => .res(+)
2	InitiatePutOD.ind \consequence =0 => .res(+)
3	PutOD.ind => .res(-) object state conflict
4	TerminatePutOD.ind => .res(-) object state conflict
5	InitiatePutOD.ind \consequence >0 => .res(-) object state conflict
6	InitiatePutOD.ind \consequence =0 => .res(-) object state conflict
7	PutOD.ind => .res(+)
8	TerminatePutOD.ind \OD not usable => .res(-) operational problem
9	TerminatePutOD.ind \OD okay => .res(+)
10	InitiatePutOD.ind \consequence >0 => .res(+)

4.3.8 EXAMPLE of a PutOD Sequence

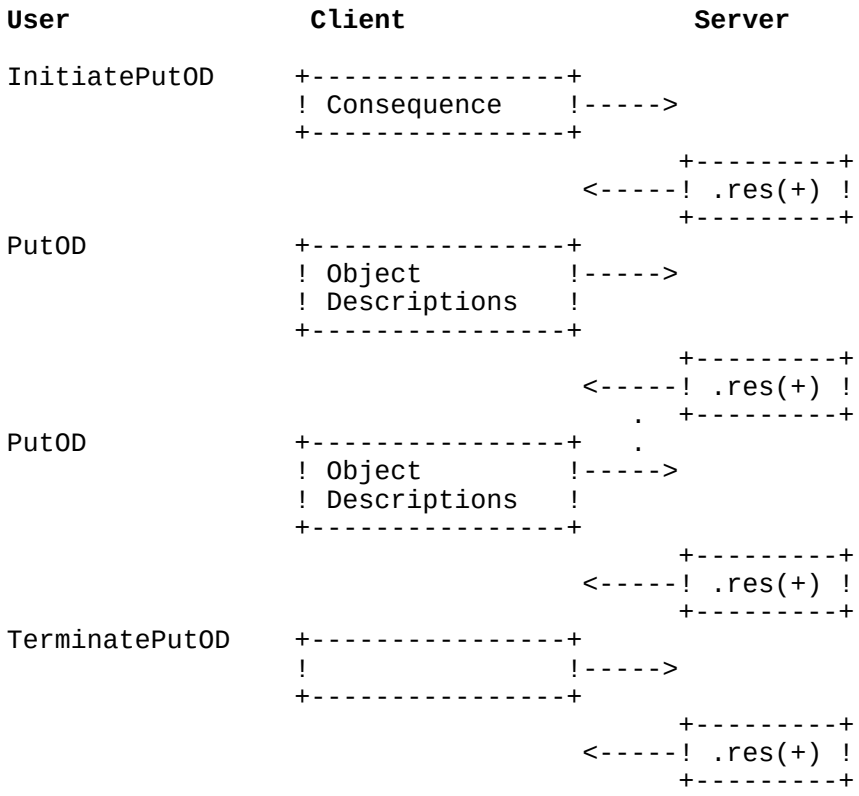


Figure 38. Example of a PutOD Sequence

4.4 Context Management

4.4.1 Model Description

The context contains all agreements concerning a communication relationship. The context management services shall be used to establish a connection, to release a connection and to reject improper services. The data necessary for this are stored in the OD object description and in the Communication Relationship List of the FMS. Transaction objects are provided for each communication relationship. The transaction objects are used for the management of confirmed service primitives.

4.4.2 The FMS CRL Object

The FMS communication relationship list (FMS CRL) contains the FMS specific descriptions of all communication relationships of a device irrespective of the time of use. The FMS CRL is organized in entries. Each entry can be addressed with the key index communication relationship (CREF). Each entry consists of a static and a dynamic part. The entry contains the complete FMS specific description of the associated communication relationship.

The FMS CRL header contains information on the structure of the FMS CRL. The FMS CRL header is addressed with CREF 0.

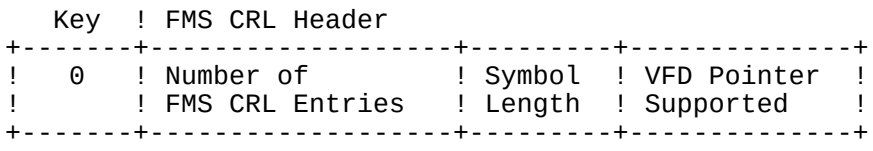


Figure 39. Structure of the FMS CRL Header

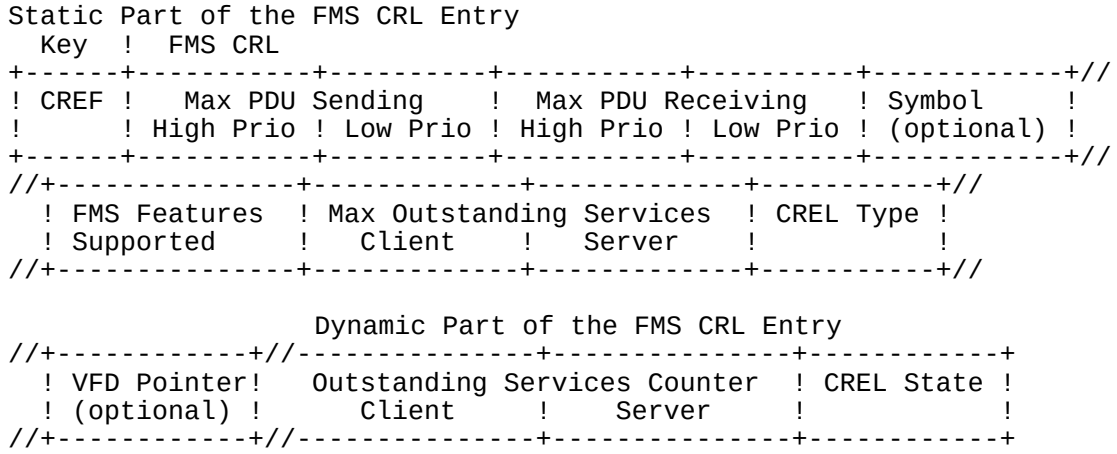


Figure 40. Structure of a FMS CRL Entry

FMS CRL Header:

Number of FMS CRL entries
This attribute shall specify the number of FMS CRL entries in the FMS CRL.

Symbol Length
This attribute shall specify the length of symbols in the FMS CRL. Only values of 0..32 are allowed.
0 <=> no symbols

VFD Pointer Supported
This attribute shall indicate, if several VFDs are sup-ported in the FMS CRL.

Static part of the FMS CRL entry:

CREF
The communication reference (CREF) is a locally unique address of the communication relationship.

Max PDU Sending High Prio
This attribute shall specify the maximum length of the FMS PDU to be sent with high priority that may be handled on this communication relationship.

Max PDU Sending Low Prio
This attribute shall specify the maximum length of the FMS PDU to be sent with low priority that may be handled on this communication relationship.

Max PDU Receiving High Prio
This attribute shall specify the maximum length of the FMS PDU to be received with high priority that may be handled on this communication relationship.

Max PDU Receiving Low Prio

This attribute shall specify the maximum length of the FMS PDU to be received with low priority that may be handled on this communication relationship.

FMS Features Supported

This attribute shall identify the optional FMS services and the options supported on this communication relationship. For each optional service there are two bits specifying the service primitives supported. For the options there are two bits specifying for which service primitives the options are permissible. The definition of the assignment of services to master/slave and to objects, and the OD object shall be observed when selecting the FMS features.

Table 13. Attribute FMS Features Supported

Service	Primitive bit[n]	Primitive bit[m]
GetOD (Long form)	.req,.con 0	.ind,.res 24
UnsolicitedStatus	.req 1	.ind 25
InitiatePutOD	.req,.con 2	.ind,.res 26
PutOD	.req,.con "	.ind,.res "
TerminatePutOD	.req,.con "	.ind,.res "
InitiateDownloadSequence	.req,.con 3	.ind,.res 27
DownloadSegment	.ind,.res "	.req,.con "
TerminateDownloadSequence	.ind,.res "	.req,.con "
InitiateUploadSequence	.req,.con 4	.ind,.res 28
UploadSegment	.req,.con "	.ind,.res "
TerminateUploadSequence	.req,.con "	.ind,.res "
RequestDomainDownload	.req,.con 5	.ind,.res 29
RequestDomainUpload	.req,.con 6	.ind,.res 30
CreateProgramInvocation	.req,.con 7	.ind,.res 31
DeleteProgramInvocation	.req,.con "	.ind,.res "
Start	.req,.con 8	.ind,.res 32
Stop	.req,.con "	.ind,.res "
Resume	.req,.con "	.ind,.res "
Reset	.req,.con "	.ind,.res "
Kill	.req,.con 9	.ind,.res 33
Read	.req,.con 10	.ind,.res 34
Write	.req,.con 11	.ind,.res 35
ReadWithType	.req,.con 12	.ind,.res 36
WriteWithType	.req,.con 13	.ind,.res 37
PhysRead	.req,.con 14	.ind,.res 38
PhysWrite	.req,.con 15	.ind,.res 39
InformationReport	.req 16	.ind 40
InformationReportWithType	.req 17	.ind 41
DefineVariableList	.req,.con 18	.ind,.res 42
DeleteVariableList	.req,.con "	.ind,.res "
EventNotification	.req 19	.ind 43
EventNotificationWithType	.req 20	.ind 44
AcknowledgeEventNotification	.req,.con 21	.ind,.res 45
AlterEventConditionMonitoring	.req,.con 22	.ind,.res 46
Options	**Primitive bit[n]**	**Primitive bit[m]**
Addressing with Name	.req 23	.ind 47

Explanation:
[n] : 0 to 23
[m] : 24 to 47

Max Outstanding Services Client
This attribute shall specify the maximum number of outstanding confirmed services allowed on this communication relationship as a client.

Max Outstanding Services Server
This attribute shall specify the maximum number of outstanding confirmed services allowed on this communication relationship as a server.

CREL Type
This attribute shall indicate the type of the communication relationship.
- true : connection-oriented
- false : connectionless

Symbol:
This attribute shall specify the symbolic name of the communication reference. Existence and length of the symbol are defined in the FMS CRL header.

VFD Pointer
This pointer shall refer to the assigned VFD.

Dynamic part of the FMS CRL entry:

Outstanding Services Counter Client (OSCC)
This attribute shall specify the number of currently outstanding confirmed services on this communication relationship as a client.

Outstanding Services Counter Server (OSCS)
This attribute shall specify the number of outstanding services that are currently being processed on this communication relationship as a server.

CREL State
This attribute shall specify the state of the communication relationship. A connection-oriented communication relationship may be in one of the following states:

- CONNECTION-NOT-ESTABLISHED
- CONNECTION-ESTABLISHING (CALLING)
- CONNECTION-ESTABLISHING (CALLED)
- CONNECTION-ESTABLISHED

A connectionless communication relationship shall always have the state CONNECTIONLESS-CLIENT or CONNECTIONLESS-SERVER (see context management state machines).

4.4.3 The Transaction Object

A Transaction Object shall be created upon receipt of a confirmed service indication primitive. The Transaction Object shall be uniquely related to the corresponding service primitive. The Transaction Object shall be deleted after the corresponding response primitive has been sent. The Transaction Object shall be uniquely identified by the combination of Invoke ID and communication reference.

The maximum number of Transaction Objects for each communication relationship of the server shall be specified by the Max Outstanding Services Server attribute in the FMS CRL.

The following services act on the Transaction Object:

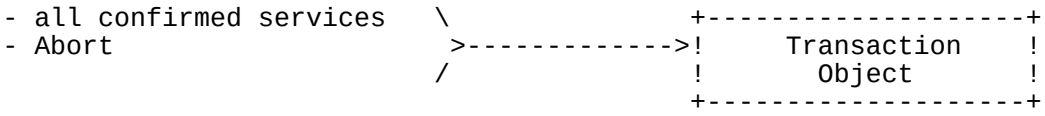


Figure 41. Transaction Object

4.4.3.1 Attributes

Object: Transaction Object
  Key Attribute: Invoke ID & CREF
  Attribute: Confirmed service indication

Invoke ID
This attribute shall identify the Transaction Object within the communication relationship.

CREF
This attribute shall identify the communication relationship on which the Transaction Object is used.

Confirmed service indication
Service identifier and argument of the pending service.

4.4.3.2 State Machine

State Machine Description

NON-EXISTENT
The Transaction Object does not exist.

PENDING
A confirmed service indication primitive has been received and the corresponding response primitive has not yet been sent.

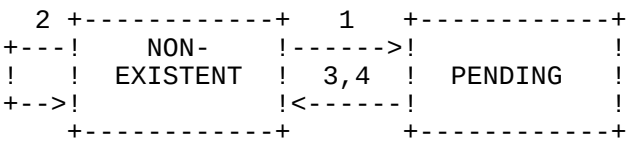


Figure 42. Transaction Object State Machine

State Transitions

1 Confirmed service.ind (Number of Transaction Objects < max)
2 Confirmed service.ind (Number of Transaction Objects = max)
3 Confirmed service.res
4 Abort.ind

4.4.4 Context Management Services

The Context Management Services comprise

- Initiate
- Abort
- Reject.

4.4.4.1 Initiate

This service shall be used to establish a connection between two communication partners and to exchange information regarding the supported services, the supported options, the maximum PDU length and the current version of the OD.

Table 14. Initiate

Parameter Name	.req	.ind	.res	.con
Argument	M	M=		
Version OD *)	M	M=		
Profile Number *)	M	M=		
Access Protection Supported *)	M	M=		
Password *)	M	M=		
Access Groups *)	M	M=		
Max PDU Sending High Prio *)				
Max PDU Sending Low Prio *)				
Max PDU Receiving High Prio *)				
Max PDU Receiving Low Prio *)				
FMS Features Supported *)				
Result(+)			S	S=
Version OD **)			M	M=
Profile Number **)			M	M=
Access Protection Supported **)			M	M=
Password **)			M	M=
Access Groups **)			M	M=
Result(-)			S	S=
Error Code			M	M
Max PDU Sending High Prio **)				M
Max PDU Sending Low Prio **)				M
Max PDU Receiving High Prio **)				M
Max PDU Receiving Low Prio **)				M
FMS Features Supported **)				M

Explanation:
*) Calling **) Called

Argument
The argument shall convey the service specific parameters of the service request.

Version OD (Calling)
This parameter shall specify the version of the client's Object Dictionary.

Profile Number (Calling)
This parameter shall specify the Profile Number of the client.

Access Protection Supported (Calling)
This parameter shall contain the Access Protection Supported attribute of the client's OD object description.

Password (Calling)
This parameter shall specify the password to be used for the access to all objects of the server on this communication relationship. If access with password is not to be used on this communication relationship, the password parameter shall have the value 0.

Access Groups (Calling)
This parameter shall specify the client's membership in specific access groups. The membership applies for the access to all objects of the server on this communication relationship.

Max PDU Sending High Prio (Calling)
This parameter shall specify the maximum length of the FMS PDU to be sent with high priority that may be handled on this communication relationship.
It shall be transmitted by the calling FMS and is not part of the primitive.

Max PDU Sending Low Prio (Calling)
This parameter shall specify the maximum length of the FMS PDU to be sent with low priority that may be handled on this communication relationship. It shall be transmitted by the calling FMS and is not part of the primitive.

Max PDU Receiving High Prio (Calling)
This parameter shall specify the maximum length of the FMS PDU to be received with high priority that may be handled on this communication relationship. It shall be transmitted by the calling FMS and is not part of the primitive.

Max PDU Receiving Low Prio (Calling)
This parameter shall specify the maximum length of the FMS PDU to be received with low priority that may be handled on this communication relationship. It shall be transmitted by the calling FMS and is not part of the primitive.

FMS Features Supported (Calling)

This parameter shall identify the optional FMS services and the options supported by the client (see FMS CRL). It shall be transmitted by the calling FMS and is not part of the primitive.

Result(+)
The Result(+) parameter shall indicate that the requested service was executed successfully.

Version OD (Called)
This parameter shall specify the version of the server's Object Dictionary.

Profile Number (Called)
This parameter shall specify the Profile Number of the server.

Access Protection Supported (Called)
This parameter shall contain the Access Protection Supported attribute of the server's OD object description.

Password (Called)
This parameter shall specify the password to be used for access to all objects of the client on this communication relationship. If access with password is not to be used on this communication relationship, the password parameter shall have the value 0.

Access Groups (Called)

This parameter shall specify the server's membership in specific access groups. The membership shall be used for the access to all objects of the client on this communication relationship.

Result(-):

The Result(-) parameter shall indicate that the service request failed.

Error Code:

This parameter shall provide the reason for the failure.

- Max PDU Size Insufficient
  The maximum PDU length is not sufficient for the communication.
- Feature Not Supported
  The requested service or option is not supported by the server.
- User Initiate Denied
  The FMS user refuses to establish the connection.
- Version OD incompatible
  The versions of the Object Dictionaries (Source OD and Remote OD) of the communication partners are not compatible.
- Password Error
  There is already a communication relationship established with the same password.
- Profile Number incompatible
  The client's profile is not supported by the server.
- Other
  Reason other than any of those identified above.

Max PDU Sending High Prio (Called)

This parameter shall specify the maximum length of the FMS PDU to be sent with high priority that may be handled on this communication relationship. It shall be transmitted by the called FMS and shall only be part of the Initiate.con primitive.

Max PDU Sending Low Prio (Called)

This parameter shall specify the maximum length of the FMS PDU to be sent with low priority that may be handled on this communication relationship. It shall be transmitted by the called FMS and shall only be part of the Initiate.con primitive.

Max PDU Receiving High Prio (Called)

This parameter shall specify the maximum length of the FMS PDU to be received with high priority that may be handled on this communication relationship. It shall be transmitted by the called FMS and shall only be part of the Initiate.con primitive.

Max PDU Receiving Low Prio (Called)

This parameter shall specify the maximum length of the FMS PDU to be received with low priority that may be handled on this communication relationship. It shall be transmitted by the called FMS and shall only be part of the Initiate.con primitive.

FMS Features Supported (Called)

This parameter shall identify the optional FMS services and the options supported by the server (see FMS CRL). It shall be transmitted by the called FMS and shall only be part of the Initiate.con primitive.

4.4.4.2 Abort

This service shall be used to release an existing communication relationship between two communication partners. Both client and server may release the connection.

Table 15. Abort

Parameter Name	.req	.ind
Argument	M	M
Locally Generated		M
Abort Identifier	M	M=
Reason Code	M	M=
Abort Detail	U	C

Argument
The argument shall convey the service specific parameters of the service request.

Locally Generated
This parameter shall indicate whether the abort was generated locally or by the communication partner.
The value false is not allowed, if the Abort Identifier parameter has the value FMS and the Reason Code parameter has the value FMS CRL Error.

Abort Identifier
This parameter shall indicate, where the reason for the abort has been detected.

0 <=> USER
1 <=> FMS
2 <=> LLI
3 <=> LAYER2

Reason Code
This parameter shall specify the reason for the abort.
If the Abort Identifier parameter has the value USER, the following values shall apply:

- ABT_RC1 <=> Disconnect
  The connection is released by the FMS user.

- ABT_RC2 <=> Version OD incompatible
  The versions of the Object Dictionaries (Source OD and Remote OD) of the communication partners are not compatible.

- ABT_RC3 <=> Password Error
  There is already a communication relationship established with the same password.

- ABT_RC4 <=> Profile Number incompatible
  The server's profile is not supported by the client.

- ABT_RC5 <=> Limited Services Permitted
  The VFD is in the Logical Status LIMITED-SERVICES-PERMITTED.

- ABT_RC6 <=> OD-loading-interacting
  The PutOD Service which is not free of interaction is currently active.

If the Abort Identifier parameter has the value FMS, the following values shall apply:

- ABT_RC1 <=> FMS CRL Error
  Faulty FMS CRL Entry

- ABT_RC2 <=> User Error
  Improper, unknown or faulty service primitive received from the FMS user

- ABT_RC3 <=> FMS PDU Error
  Unknown or faulty FMS PDU received from the LLI

- ABT_RC4 <=> Connection State Conflict LLI
  Improper LLI service primitive

- ABT_RC5 <=> LLI Error
  Unknown or faulty LLI service primitive

- ABT_RC6 <=> PDU Size
  PDU length exceeds maximum PDU length

- ABT_RC7 <=> Feature Not Supported
  SERVICE_REQ_PDU received from the LLI and service or option not supported as a server (see FMS Features Supported attribute in the FMS CRL)

- ABT_RC8 <=> Invoke ID Error Response
  Confirmed service.res received from the FMS user and Invoke ID does not exist or CONFIRMED-SERVICE_RES_PDU received from the LLI and Invoke ID does not exist

- ABT_RC9 <=> Max Services Overflow
  CONFIRMED-SERVICE_REQ_PDU received from the LLI and Outstanding Services Counter Server ≥ Outstanding Services Server

- ABT_RC10 <=> Connection State Conflict FMS
  INITIATE_REQ_PDU or INITIATE_RES_PDU received from the LLI

- ABT_RC11 <=> Service Error
  The service in the response does not match the service in the indication
  or
  the service in the confirmation does not match the service in the request

- ABT_RC12 <=> Invoke ID Error Request
  CONFIRMED-SERVICE_REQ_PDU received from the LLI and Invoke ID already exists.

- ABT_RC13 <=> FMS disabled
  FMS is not ready for data transmission.

If the Abort Identifier parameter has the value LLI or LAYER2, the Reason Code parameter shall be supplied by the LLI.

Abort Detail

This parameter contains additional information about the abort reason (max. 16 octets). In case of error reports from the application, the meaning is defined in the profile.

4.4.4.3 Reject

The Reject service is used by the FMS to reject an improper PDU or an improper FMS-Service_Request or _Response from the FMS user.

Table 16. Reject

Parameter Name	.ind
Argument	
Detected Here	M
Original Invoke ID	C
Reject PDU Type	M
Reject Code	M

Argument
The argument shall convey the service specific parameters of the service request.

Detected Here
This parameter shall indicate if the error has been detected locally (true). The value false is allowed only if the Reject PDU Type parameter has the value Confirmed-Response-PDU and the Reject Code parameter has the value PDU Size.

Original Invoke ID
The original Invoke ID of the rejected PDU or the rejected FMS Service.

Reject PDU Type
This parameter shall indicate the type of the rejected PDU or of the PDU which would have been resulted from the rejected FMS Service. The permissible values are as follows:

1 <=> Confirmed-Request-PDU
2 <=> Confirmed-Response-PDU
3 <=> Unconfirmed-PDU
4 <=> Unknown type of PDU

Reject Code
This parameter shall secify the reason for the reject.

1 <=> Invoke-ID-Exists
Confirmed service.req received from the FMS user and Invoke ID already exists

2 <=> Max-Services-Overflow
Confirmed service.req received from the FMS user and Out-standing Services Counter Client ≥ Outstanding Services Client

3 <=> Feature-Not-Supported-Connection-Oriented
Service.req received from the FMS user and service or option not supported as a client (see FMS Features Supported attribute in the FMS CRL)

4 <=> Feature-Not-Supported-Connectionless
Unconfirmed service.req received from the FMS user and service or option not supported as a client (see FMS Features Supported attribute in the FMS CRL) or confirmed service.req received from the FMS user

5 <=> PDU-Size
PDU length exceeds maximum PDU length

6 <=> User-Error-Connectionless
Improper or faulty service primitive received from the FMS user

0 <=> Other
Reason other than any of those identified above.

4.4.5 Tests at Connection Establishment

4.4.5.1 Context Test in FMS

Upon receipt of an INITIATE_REQ_PDU the server's FMS shall check the FMS Context of the communication partner (remote Context), if it is compatible with its own FMS Context (local Context) as defined for this connection in the FMS CRL. The local FMS Context is assumed to be correct. The compatibility of the local with the remote context is defined by the following table (see figure below). In this table, meaningless fields are left blank (e.g. the combination of local Context "Max PDU Sending High Prio" with remote Context "FMS Features Supported"). Those combinations shall not be checked.

	local Context					
	Max PDU				FMS Features	
remote Context	Sending		Receiving		Supported	
	High	Low	High	Low	.req	.ind
	Prio	Prio	Prio	Prio	[n]	[m]
					0 1	0 1
Max PDU Sending High Prio			≥			
Max PDU Sending Low Prio				≥		
Max PDU Receiving High Prio	≤					
Max PDU Receiving Low Prio		≤				
FMS Features Supported .req [n] 0						X X
FMS Features Supported .req [n] 1						- X
FMS Features Supported .ind [m] 0					X -	
FMS Features Supported .ind [m] 0					X X	

Explanation:
- .req 0 : Feature is not used as Client
- .req 1 : Feature is used as Client
- .ind 0 : Feature is not supported as Server
- .ind 1 : Feature is supported as Server
- ≤ : local value smaller than or equal remote value
- ≥ : local value larger than or equal remote value
- X : compatible
- - : not compatible (error case)
- [n] : 0 to 23 [m] : 24 to 47

Figure 43. Compatibility of the local FMS Context to the remote FMS Context

4.4.5.2 Tests at the FMS User

- **Password Test**
  If the FMS user supports access with password, it shall check if the password is unambiguous. That means, if there is a password, it shall be different from the passwords of all other communication relationships.

- If the password is not unambiguous and the FMS user is the server of the Initiate service, it shall issue an Initiate.response primitive with the Result(-) parameter and with Error Code "Password Error".

- If the password is not unambiguous and the FMS user is the client of the Initiate service, it shall release the connection with the Abort service, with Reason Code "Password Error".

- Test of Version OD

If the FMS user has a remote OD for this connection, it may check if the received parameter "Version OD" is compatible with the attribute "Version OD" of the associated remote OD.

- If they are incompatible and the FMS user is the server of the Initiate service, it may issue an Initiate.response primitive with the Result(-) parameter and with Error Code "Version OD incompatible".
- If they are incompatible and the FMS user is the client of the Initiate service, it may release the connection with the Abort service, with Reason Code "Version OD incompatible".

- Test of Profile Number

The FMS user may check if the received parameter "Profile Number" is compatible with the attribute "Profile Number" of the VFD.

- If they are incompatible and the FMS user is the server of the Initiate service, it may issue an Initiate.response primitive with the Result(-) parameter and with Error Code "Profile Number incompatible".
- If they are incompatible and the FMS user is the client of the Initiate service, it may release the connection with the Abort service, with Reason Code "Profile Number incompatible".

4.4.6 State Machine for connection-oriented Communication Relationships

4.4.6.1 State Machine Description

CONNECTION-NOT-ESTABLISHED

The connection is not established. Only the service primitives Initiate.req, ASS.ind, Abort.req and ABT.ind are allowed. All other services shall be rejected with the Abort service.

CONNECTION-ESTABLISHING (CALLING)

The local FMS user wishes to establish the connection. Only the service primitives ASS.con(+), ASS.con(-), Abort.req and ABT.ind are allowed. All other services shall be rejected with the Abort service. This state is abbreviated as CON-ESTABLISHING-CALLING.

CONNECTION-ESTABLISHING (CALLED)

The remote FMS user wishes to establish the connection. Only the service primitives Initiate.res(+), Initiate.res(-), Abort.req and ABT.ind are allowed. All other services shall be rejected with the Abort service. This state is abbreviated as CON-ESTABLISHING-CALLED.

CONNECTION-ESTABLISHED

The communication relationship is established. The service primitives Initiate.req, Initiate.res(+), Initiate.res(-), DTU.ind are not allowed and shall be rejected with the Abort service.

The following actions shall be taken to reset a communication reference (Reset CREF) :

- Clear memory contents
- Set attribute "Outstanding Services Counter Client" and attribute "Outstanding Services Counter Server" of the FMS CRL (dynamic part) to 0
- Set state of the communication relationship to "CONNECTION-NOT-ESTABLISHED"

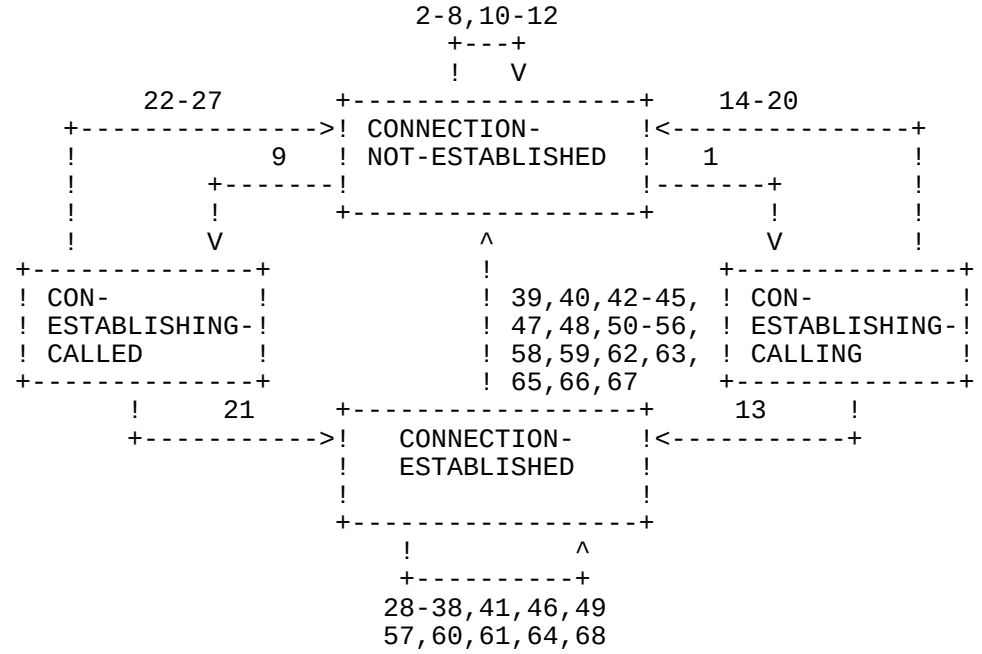


Figure 44. State Machine

4.4.6.2 State Transitions

State Transitions at Connection Establishment / Release:

Current State / Event / \Exit Condition / => Action Taken	Transition	Next State
CONNECTION-NOT-ESTABLISHED Initiate.req from FMS user received \FMS CRL entry valid => send INITIATE_REQ_PDU, (ASS.req) to LLI	**1**	**CON-ESTABLISHING-CALLING**
CONNECTION-NOT-ESTABLISHED Initiate.req from FMS user received \FMS CRL entry invalid => Abort.ind to FMS user <reason code = ABT_RC1>	**2**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED Abort.req received from FMS user => ignore	**3**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED not allowed, unknown or faulty service primitive received from FMS user => Abort.ind to FMS user <reason code = ABT_RC2>	**4**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED ABT.ind received from LLI => ignore	**5**	**CONNECTION-NOT-ESTABLISHED**

Current State Event \Exit Condition => Action Taken	Transition	Next State
CONNECTION-NOT-ESTABLISHED faulty or unknown service primitive received from LLI => ABT.req to LLI <reason code = ABT_RC5>	**6**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED not allowed LLI service primitive received from LLI => ABT.req to LLI <reason code = ABT_RC4>	**7**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED not allowed, unknown or faulty FMS PDU (ASS.ind) received => ABT.req to LLI <reason code = ABT_RC3>	**8**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED INITIATE_REQ_PDU(ASS.ind from LLI) \FMS CRL entry is valid AND FMS context test is positive => Initiate.ind to FMS user	**9**	**CON-ESTABLISHING-CALLED**
CONNECTION-NOT-ESTABLISHED INITIATE_REQ_PDU(ASS.ind from LLI) \FMS CRL entry is invalid => ABT.req to LLI <reason code = ABT_RC1>	**10**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED INITIATE_REQ_PDU (ASS.ind) from LLI \FMS CRL entry is valid AND max PDU length test is negative => send INITIATE_ERROR_PDU (ASS.res(-)) to LLI <error code = 1>	**11**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED INITIATE_REQ_PDU (ASS.ind) from LLI \FMS CRL entry is valid AND max PDU length test is positive AND features supported test is negative => send INITIATE_ERROR_PDU (ASS.res(-)) to LLI <error code = 2>	**12**	**CONNECTION-NOT-ESTABLISHED**
CON-ESTABLISHING-CALLING INITIATE_RES_PDU (ASS.con(+)) from LLI => Initiate.con(+) to FMS user	**13**	**CONNECTION-ESTABLISHED**
CON-ESTABLISHING-CALLING INITIATE_ERROR_PDU (ASS.con(-)) from LLI => Initiate.con(-) to FMS user reset CREF	**14**	**CONNECTION-NOT-ESTABLISHED**
CON-ESTABLISHING-CALLING ABT.ind received from LLI => Abort.ind to FMS user <reason code out of ABT.ind> reset CREF	**15**	**CONNECTION-NOT-ESTABLISHED**

Current State	Transition	Next State
Event \Exit Condition => Action Taken		

CON-ESTABLISHING-CALLING **16** **CONNECTION-NOT-ESTABLISHED**
Abort.req received from FMS user
=> ABT.req to LLI <reason code as given by user>
reset CREF

CON-ESTABLISHING-CALLING **17** **CONNECTION-NOT-ESTABLISHED**
faulty or unknown service primitive received from LLI
=> ABT.req to LLI <reason code = ABT_RC5>
Abort.ind to FMS user <reason code = ABT_RC5>
reset CREF

CON-ESTABLISHING-CALLING **18** **CONNECTION-NOT-ESTABLISHED**
not allowed LLI service primitive received from LLI
=> ABT.req to LLI <reason code = ABT_RC4>
Abort.ind to FMS user <reason code = ABT_RC4>
reset CREF

CON-ESTABLISHING-CALLING **19** **CONNECTION-NOT-ESTABLISHED**
not allowed, unknown or faulty FMS PDU (ASS.con)
received from LLI
=> ABT.req to LLI <reason code = ABT_RC3>
Abort.ind to FMS user <reason code = ABT_RC3>
reset CREF

CON-ESTABLISHING-CALLING **20** **CONNECTION-NOT-ESTABLISHED**
not allowed, unknown or faulty service
primitive received from FMS user
=> ABT.req to LLI <reason code = ABT_RC2>
Abort.ind to FMS user <reason code = ABT_RC2>
reset CREF

CON-ESTABLISHING-CALLED **21** **CONNECTION-ESTABLISHED**
Initiate.res(+) received from FMS user
=> send INITIATE_RES_PDU (ASS.res(+)) to LLI

CON-ESTABLISHING-CALLED **22** **CONNECTION-NOT-ESTABLISHED**
Initiate.res(-) received from FMS user
=> send INITIATE_ERROR_PDU (ASS.res(-)) to LLI
reset CREF

CON-ESTABLISHING-CALLED **23** **CONNECTION-NOT-ESTABLISHED**
Abort.req received from FMS user
=> ABT.req to LLI <reason code as given by user>
reset CREF

CON-ESTABLISHING-CALLED **24** **CONNECTION-NOT-ESTABLISHED**
ABT.ind received from LLI
=> Abort.ind to FMS user <reason code out of ABT.ind>
reset CREF

CON-ESTABLISHING-CALLED **25** **CONNECTION-NOT-ESTABLISHED**
faulty or unknown service primitive
received from LLI
=> ABT.req to LLI <reason code = ABT_RC5>
Abort.ind to FMS user <reason code = ABT_RC5>
reset CREF

Current State Event \Exit Condition => Action Taken	Transition	Next State
CON-ESTABLISHING-CALLED not allowed LLI service primitive received from LLI => ABT.req to LLI <reason code = ABT_RC4> Abort.ind to FMS user <reason code = ABT_RC4>, reset CREF	**26**	**CONNECTION-NOT-ESTABLISHED**
CON-ESTABLISHING-CALLED not allowed, unknown or faulty service primitive received from FMS user => ABT.req to LLI <reason code = ABT_RC2> Abort.ind to FMS user <reason code = ABT_RC2>, reset CREF	**27**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED confirmed Service.req received from FMS user \OSCC < max outstanding services client AND invoke ID not existent AND PDU length ≤ max PDU sending low prio AND features supported test (Client) positive => confirmed service_REQ_PDU with DTC.req to LLI OSCC := OSCC + 1	**28**	**CONNECTION-ESTABLISHED**
CONNECTION-ESTABLISHED confirmed Service.req received from FMS user \OSCC ≥ max outstanding services client => Reject.ind to FMS user <reject code = 2>	**29**	**CONNECTION-ESTABLISHED**
CONNECTION-ESTABLISHED confirmed Service.req received from FMS user \OSCC < max outstanding services client AND invoke ID already existent => Reject.ind to FMS user <reject code = 1>	**30**	**CONNECTION-ESTABLISHED**
CONNECTION-ESTABLISHED confirmed Service.req received from FMS user \OSCC < max outstanding services client AND invoke ID not existent AND PDU length > max PDU sending low prio => Reject.ind to FMS user <reject code = 5>	**31**	**CONNECTION-ESTABLISHED**
CONNECTION-ESTABLISHED confirmed Service.req received from FMS user \OSCC < max outstanding services client AND invoke ID not existent AND PDU length ≤ max PDU sending low prio AND features supported test (Client) negative => Reject.ind to FMS user <reject code = 3>	**32**	**CONNECTION-ESTABLISHED**
CONNECTION-ESTABLISHED unconfirmed Service.req <priority = false> received from FMS user \PDU length ≤ max PDU sending low prio AND features supported test (Client) positive => unconfirmed Service_REQ_PDU (DTA.req <priority = low>) to LLI	**33**	**CONNECTION-ESTABLISHED**

Current State Event \Exit Condition => Action Taken	Transition	Next State
CONNECTION-ESTABLISHED unconfirmed Service.req <priority = false> received from FMS user \PDU length > max PDU sending low prio => Reject.ind to FMS user <reject code = 5>	**34**	**CONNECTION-ESTABLISHED**
CONNECTION-ESTABLISHED unconfirmed Service.req <priority = false> received from FMS user \PDU length ≤ max PDU sending low prio AND features supported test (Client) negative => Reject.ind to FMS user <reject code = 3>	**35**	**CONNECTION-ESTABLISHED**
CONNECTION-ESTABLISHED unconfirmed Service.req <priority = true> received from FMS user \PDU length ≤ max PDU sending high prio AND features supported test (Client) positive => unconfirmed Service_REQ_PDU (DTA.req <priority = high>) to LLI	**36**	**CONNECTION-ESTABLISHED**
CONNECTION-ESTABLISHED unconfirmed Service.req <priority = true> received from FMS user \PDU length > max PDU sending high prio => Reject.ind to FMS user <reject code = 5>	**37**	**CONNECTION-ESTABLISHED**
CONNECTION-ESTABLISHED unconfirmed Service.req <priority = true> received from FMS user \PDU length ≤ max PDU sending high prio AND features supported test (Client) negative => Reject.ind to FMS user <reject code = 3>	**38**	**CONNECTION-ESTABLISHED**
CONNECTION-ESTABLISHED Abort.req received from FMS user => ABT.req to LLI <reason code as given by user> reset CREF	**39**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED faulty, unknown or not allowed service primitive received from FMS user => ABT.req to LLI <reason code = ABT_RC2> Abort.ind to FMS user <reason code = ABT_RC2> reset CREF	**40**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED confirmed Service_REQ_PDU (DTC.ind) from LLI \PDU length ≤ max PDU receiving low prio AND OSCS < max outstanding services server AND invoke ID not existent AND features supported test (Server) positive => confirmed Service.ind to FMS user OSCS := OSCS + 1	**41**	**CONNECTION-ESTABLISHED**

Current State Event \Exit Condition => Action Taken	Transition	Next State
CONNECTION-ESTABLISHED confirmed Service_REQ_PDU (DTC.ind) from LLI \PDU length > max PDU receiving low prio => ABT.req to LLI <reason code = ABT_RC6> Abort.ind to FMS user <reason code = ABT_RC6> reset CREF	**42**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED confirmed Service_REQ_PDU (DTC.ind) from LLI \PDU length ≤ max PDU receiving low prio AND OSCS ≥ max outstanding services server => ABT.req to LLI <reason code = ABT_RC9> Abort.ind to FMS user <reason code = ABT_RC9> reset CREF	**43**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED confirmed Service_REQ_PDU (DTC.ind) from LLI \PDU length ≤ max PDU receiving low prio AND OSCS < max outstanding services server AND invoke ID already existent => ABT.req to LLI <reason code = ABT_RC12> Abort.ind to FMS <reason code = ABT_RC12> reset CREF	**44**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED confirmed Service_REQ_PDU (DTC.ind) from LLI \PDU length ≤ max PDU receiving low prio AND OSCS < max outstanding services server AND invoke ID not existent AND features supported test (Server) negative => ABT.req to LLI <reason code = ABT_RC7> Abort.ind to FMS user <reason code = ABT_RC7> reset CREF	**45**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED unconfirmed Service_REQ_PDU <priority = low> (DTA.ind) from LLI \PDU length ≤ max PDU receiving low prio AND features supported test (Server) positive => unconfirmed Service.ind <priority = false> to FMS user	**46**	**CONNECTION-ESTABLISHED**
CONNECTION-ESTABLISHED unconfirmed Service_REQ_PDU <priority = low> (DTA.ind) from LLI \PDU length > max PDU receiving low prio => ABT.req to LLI <reason code = ABT_RC6> Abort.ind to FMS user <reason code = ABT_RC6> reset CREF	**47**	**CONNECTION-NOT-ESTABLISHED**

Current State	Transition	Next State
Event \Exit Condition => Action Taken		

CONNECTION-ESTABLISHED **48** **CONNECTION-NOT-ESTABLISHED**

unconfirmed Service_REQ_PDU <priority = low>
(DTA.ind) from LLI

\PDU length ≤ max PDU receiving low prio
AND features supported test (Server) negative

=> ABT.req to LLI <reason code = ABT_RC7>
Abort.ind to FMS user <reason code = ABT_RC7>
reset CREF

CONNECTION-ESTABLISHED **49** **CONNECTION-ESTABLISHED**

unconfirmed Service_REQ_PDU <priority = high>
(DTA.ind) from LLI

\PDU length ≤ max PDU receiving high prio
AND features supported test (Server) positive

=> unconfirmed Service.ind <priority = true> to FMS user

CONNECTION-ESTABLISHED **50** **CONNECTION-NOT-ESTABLISHED**

unconfirmed Service_REQ_PDU <priority = high>
(DTA.ind) from LLI

\PDU length > max PDU receiving high prio

=> ABT.req to LLI <reason code = ABT_RC6>
Abort.ind to FMS user <reason code = ABT_RC6>
reset CREF

CONNECTION-ESTABLISHED **51** **CONNECTION-NOT-ESTABLISHED**

unconfirmed Service_REQ_PDU <priority = high>
(DTA.ind) from LLI

\PDU length ≤ max PDU receiving high prio
AND features supported test (Server) negative

=> ABT.req to LLI <reason code = ABT_RC7>
Abort.ind to FMS user <reason code = ABT_RC7>
reset CREF

CONNECTION-ESTABLISHED **52** **CONNECTION-NOT-ESTABLISHED**

ABT.ind from LLI received

=> Abort.ind to FMS user <reason code out of ABT.ind>
reset CREF

CONNECTION-ESTABLISHED **53** **CONNECTION-NOT-ESTABLISHED**

INITIATE_REQ_PDU (ASS.ind) or INITIATE_RES_PDU (ASS.con)
from LLI

=> ABT.req to LLI <reason code = ABT_RC10>
Abort.ind to FMS user <reason code = ABT_RC10>
reset CREF

CONNECTION-ESTABLISHED **54** **CONNECTION-NOT-ESTABLISHED**

faulty or unknown service primitive from LLI received

=> ABT.req to LLI <reason code = ABT_RC5>
Abort.ind to FMS user <reason code = ABT_RC5>
reset CREF

Current State / Event / \Exit Condition / => Action Taken	Transition	Next State

CONNECTION-ESTABLISHED **55** **CONNECTION-NOT-ESTABLISHED**

Event: not allowed LLI service primitive received from LLI

=> ABT.req to LLI <reason code = ABT_RC4>
Abort.ind to FMS user <reason code = ABT_RC4>
reset CREF

CONNECTION-ESTABLISHED **56** **CONNECTION-NOT-ESTABLISHED**

Event: not allowed, unknown or faulty FMS PDU received from LLI

=> ABT.req to LLI <reason code = ABT_RC3>
Abort.ind to FMS user <reason code = ABT_RC3>
reset CREF

CONNECTION-ESTABLISHED **57** **CONNECTION-ESTABLISHED**

Event: confirmed Service.res (+) received from FMS user

\Invoke ID existent as server
AND service out of .res identical with
service out of .ind
AND PDU length ≤ max PDU sending low prio

=> confirmed Service_RES_PDU with DTC.res to LLI
OSCS := OSCS - 1

CONNECTION-ESTABLISHED **68** **CONNECTION-ESTABLISHED**

Event: confirmed Service.res (-) received from FMS user

\Invoke ID existent as server
AND service out of .res identical with
service out of .ind
AND PDU length ≤ max PDU sending low prio

=> confirmed Error_PDU with DTC.res to LLI
OSCS := OSCS - 1

CONNECTION-ESTABLISHED **58** **CONNECTION-NOT-ESTABLISHED**

Event: confirmed Service.res received from FMS user

\Invoke ID not existent as server

=> ABT.req to LLI <reason code = ABT_RC8>
Abort.ind to FMS user <reason code = ABT_RC8>
reset CREF

CONNECTION-ESTABLISHED **59** **CONNECTION-NOT-ESTABLISHED**

Event: confirmed Service.res received from FMS user

\Invoke ID existent as server
AND service out of .res not identical with
service out of .ind

=> ABT.req to LLI <reason code = ABT_RC11>
Abort.ind to FMS user <reason code = ABT_RC11>
reset CREF

CONNECTION-ESTABLISHED **60** **CONNECTION-ESTABLISHED**

Event: confirmed Service.res received from FMS user

\Invoke ID existent as server
AND service out of .res identical with
Service out of .ind
AND PDU length > max PDU sending low prio

=> REJECT_PDU with DTC.res to LLI <reject code = 5>
OSCS := OSCS - 1

Current State	Transition	Next State
Event \Exit Condition => Action Taken		

CONNECTION-ESTABLISHED **61** **CONNECTION-ESTABLISHED**

confirmed Service_RES_PDU or confirmed Error_PDU (DTC.con)from LLI

\PDU length ≤ max PDU receiving low prio
AND invoke ID existent as client
AND service out of .con identical with
service out of .req

=> confirmed Service.con to FMS user, OSCC := OSCC - 1

CONNECTION-ESTABLISHED **62** **CONNECTION-NOT-ESTABLISHED**

confirmed Service_RES_PDU or confirmed Error_PDU (DTC.con) from LLI

\PDU length > max PDU receiving low prio

=> ABT.req to LLI <reason code = ABT_RC6>
Abort.ind to FMS user <reason code = ABT_RC6>
reset CREF

CONNECTION-ESTABLISHED **63** **CONNECTION-NOT-ESTABLISHED**

confirmed Service_RES_PDU or confirmed Error_PDU (DTC.con) from LLI

\PDU length ≤ max PDU receiving low prio
AND invoke ID not existent as client

=> ABT.req to LLI <reason code = ABT_RC8>
Abort.ind to FMS user <reason code = ABT_RC8>
reset CREF

CONNECTION-ESTABLISHED **67** **CONNECTION-NOT-ESTABLISHED**

confirmed Service_RES_PDU or confirmed Error_PDU (DTC.con) from LLI

\PDU length ≤ max PDU receiving low prio
AND invoke ID existent as client
AND service out of .con not identical with
service out of .req

=> ABT.req to LLI <reason code = ABT RC11>
Abort.ind to FMS user <reason code = ABT RC11>
Reset CREF

CONNECTION-ESTABLISHED **64** **CONNECTION-ESTABLISHED**

REJECT_PDU (DTC.con) from LLI

\Original invoke ID existent as client
AND (<reject code = 5>
OR <reject code = 0>)

=> Reject.ind to FMS user <reject code = 5>
OSCC := OSCC - 1

CONNECTION-ESTABLISHED **65** **CONNECTION-NOT-ESTABLISHED**

REJECT_PDU (DTC.con) from LLI

\Original invoke ID not existent as client

=> ABT.req to LLI <reason code = ABT_RC8>
Abort.ind to FMS user <reason code = ABT_RC8>
reset CREF

CONNECTION-ESTABLISHED **66** **CONNECTION-NOT-ESTABLISHED**

REJECT_PDU (DTC.con) from LLI

\Original invoke ID existent as client
AND (<reject code unequal 5>
AND <reject code unequal 0>)

=> ABT.req to LLI <reason code = ABT_RC3>
Abort.ind to FMS user <reason code = ABT_RC3>
reset CREF

4.4.7 State Machine for connectionless Communication Relationships

4.4.7.1 State Machine in the Client

State Machine Description

CONNECTIONLESS-CLIENT
The communication relationship is ready for operation.
Only an unconfirmed service.req primitive is permitted, all other services shall be rejected or ignored.

Figure 45. State Machine

4.4.7.1.1 State Transitions

Current State Event \Exit Condition => Action Taken	Transition	Next State
CONNECTIONLESS-CLIENT unconfirmed Service.req <priority = false) received from FMS user \PDU length ≤ max PDU sending low prio AND features supported test (Client) positive => unconfirmed Service_REQ_PDU (DTU.req <priority = low>) to LLI	**1**	**CONNECTIONLESS-CLIENT**
CONNECTIONLESS-CLIENT unconfirmed Service.req <priority = false) received from FMS user \PDU length > max PDU sending low prio => Reject.ind to FMS user <reject code = 5>	**2**	**CONNECTIONLESS-CLIENT**
CONNECTIONLESS-CLIENT unconfirmed Service.req <priority = false) received from FMS user \PDU length ≤ max PDU sending low prio AND features supported test (Client) negative => Reject.ind to FMS user <reject code = 4>	**3**	**CONNECTIONLESS-CLIENT**
CONNECTIONLESS-CLIENT unconfirmed Service.req <priority = true) received from FMS user \PDU length ≤ max PDU sending high prio AND features supported test (Client) positive => unconfirmed Service_REQ_PDU (DTU.req <priority = high>) to LLI	**4**	**CONNECTIONLESS-CLIENT**

Current State Event \Exit Condition => Action Taken	Transition	Next State
CONNECTIONLESS-CLIENT unconfirmed Service.req <priority = true) received from FMS user \PDU length > max PDU sending high prio => Reject.ind to FMS user <reject code = 5>	**5**	**CONNECTIONLESS-CLIENT**
CONNECTIONLESS-CLIENT unconfirmed Service.req <priority = true> received from FMS user \PDU length ≤ max PDU sending high prio AND features supported test (Client) negative => Reject.ind to FMS user <reject code = 4>	**6**	**CONNECTIONLESS-CLIENT**
CONNECTIONLESS-CLIENT confirmed Service.req received from FMS user => Reject.ind to FMS user <reject code = 4>	**7**	**CONNECTIONLESS-CLIENT**
CONNECTIONLESS-CLIENT not allowed, unknown or faulty service primitive received from FMS user => Reject.ind to FMS user <reject code = 6>	**8**	**CONNECTIONLESS-CLIENT**
CONNECTIONLESS-CLIENT LLI service primitive received => ignore	**9**	**CONNECTIONLESS-CLIENT**

4.4.7.2 State Machine in the Server

4.4.7.2.1 State Machine Description

CONNECTIONLESS-SERVER

The communication relationship is ready for operation.

Only DTU.ind from LLI is permitted, other services shall be rejected or ignored.

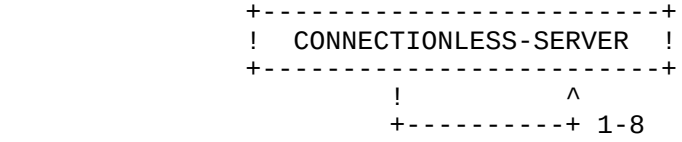


Figure 46. State Machine

State Transitions

Current State Event \Exit Condition => Action Taken	Transition	Next State
CONNECTIONLESS-SERVER unconfirmed Service_REQ_PDU (DTU.ind <priority = low>) from LLI \PDU length ≤ max PDU receiving low prio AND features supported test positive => unconfirmed Service.ind <priority = false> to FMS user	**1**	**CONNECTIONLESS-SERVER**
CONNECTIONLESS-SERVER unconfirmed Service_REQ_PDU (DTU.ind <priority = low>) from LLI \PDU length > max PDU receiving low prio => ignore	**2**	**CONNECTIONLESS-SERVER**
CONNECTIONLESS-SERVER unconfirmed Service_REQ_PDU (DTU.ind <priority = low>) from LLI \PDU length ≤ max PDU receiving low prio AND features supported test negative => ignore	**3**	**CONNECTIONLESS-SERVER**
CONNECTIONLESS-SERVER unconfirmed Service_REQ_PDU (DTU.ind <priority = high>) from LLI \PDU length ≤ max PDU receiving high prio AND features supported test positive => unconfirmed Service.ind <priority = true> to FMS user	**4**	**CONNECTIONLESS-SERVER**
CONNECTIONLESS-SERVER unconfirmed Service_REQ_PDU (DTU.ind <priority = high>) from LLI \PDU length > max PDU receiving high prio => ignore	**5**	**CONNECTIONLESS-SERVER**
CONNECTIONLESS-SERVER unconfirmed Service_REQ_PDU (DTU.ind <priority = high>) from LLI \PDU length ≤ max PDU receiving high prio AND features supported test negative => ignore	**6**	**CONNECTIONLESS-SERVER**
CONNECTIONLESS-SERVER not allowed, unknown or faulty LLI service primitive or not allowed, unknown or faulty FMS PDU received => ignore	**7**	**CONNECTIONLESS-SERVER**
CONNECTIONLESS-SERVER service primitive received from FMS user => Reject.ind to FMS user <reject code = 6>	**8**	**CONNECTIONLESS-SERVER**

4.5 Domain Management

4.5.1 Model Description

A Domain is a part of memory. It may contain programs or data. The Domain shall be of data type "octet string". The maximum number of octets of a Domain shall be defined in the Object Description. Only one Download service or one Upload service, but not both, may operate on a Domain at any one time.

The following services operate on Domain Objects:

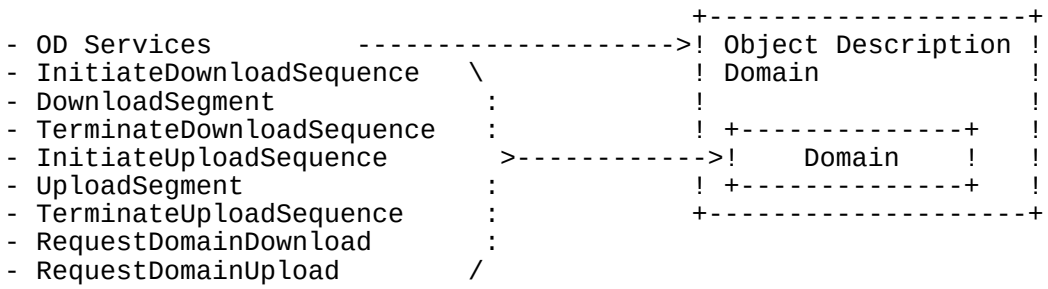


Figure 47. Domain Services

4.5.2 The Domain Object

4.5.2.1 Attributes

```
Object:  Domain
  Key Attribute:  Index
  Key Attribute:  Domain Name
  Attribute:  Max Octets
  Attribute:  Password
  Attribute:  Access Groups
  Attribute:  Access Rights
  Attribute:  Local Address
  Attribute:  Domain State
  Attribute:  Upload State
  Attribute:  Counter
  Attribute:  Extension
```

Index
Logical address of the object in OD.

Domain Name
This attribute specifies the symbolic name of the Domain. Existence and length are defined in the OD Object Description.

Max Octets
This attribute specifies the max. number of octets of the Domain.

Password
This attribute shall specify the password for the access rights.

Access Groups
This attribute shall specify the object's membership in specific access groups. The object is member of an access group if the corresponding bit is set.

Table 17. Access Groups for a Domain

Bit	Meaning
7	Access Group 1
6	Access Group 2
5	Access Group 3
4	Access Group 4
3	Access Group 5
2	Access Group 6
1	Access Group 7
0	Access Group 8

Access Rights

This attribute shall specify the rights to access the object. The respective access is allowed if the corresponding bit is set.

Table 18. Access Rights for Domains

Bit	Name	Meaning
7	R	Right to Read the registered Password
6	W	Right to Write the registered Password
5	U	Right to Use in a PI for the registered Password
3	Rg	Right to Read for Access Groups
2	Wg	Right to Write for Access Groups
1	Ug	Right to Use in a PI for Access Groups
15	Ra	Right to Read for all Communication Partners
14	Wa	Right to Write for all Communication Partners
13	Ua	Right to Use in a PI for all Communication Partners

Local Address

This attribute is a system specific reference to the real object. It may be used internally for addressing the object. If a reference of this kind is not applied, the Local Address attribute shall have the value FFFFFFFF hex.

Domain State

This attribute shall specify the state of the Domain Object.

1 <=> EXISTENT 2 <=> LOADING 3 <=> INCOMPLETE
4 <=> COMPLETE 5 <=> READY 6 <=> IN-USE

Upload State

This attribute shall specify the state of the State Machine for Upload of the Domain.

0 <=> NON-EXISTENT
1 <=> UPLOADING
2 <=> UPLOADED

Counter

This attribute shall specify the number of Program Invocations which currently use this Domain. The Program Invocation Management maintains this Counter, while the Domain Management only interprets the Counter. If the Counter attribute has a value greater than 0, the Domain is in use and may not be overwritten with a Download service.

Extension

This attribute contains profile specific information.

4.5.2.2 Object Description of the OD on Transmission

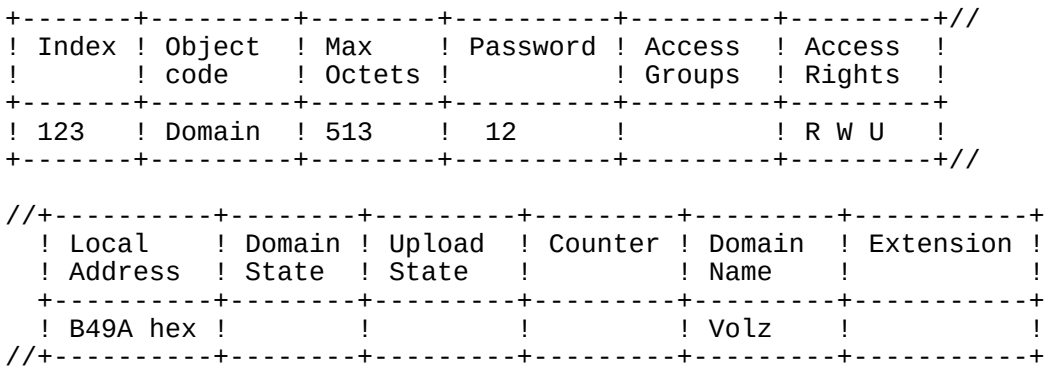


Figure 48. Structure of an Entry in the S-OD

Objektcode
Identification of a Domain Object. This identification shall be transmitted in the GetOD service and in the PutOD service in addition to the object attributes.

4.5.3 Download Services

The Download Services shall be used to load data from the client into the server's Domain.

4.5.3.1 InitiateDownloadSequence

The InitiateDownloadSequence service shall be used to begin the loading of the Domain whose Index or Domain Name is included in the service request.

Table 19. InitiateDownloadSequence

Parameter Name	.req	.res
	.ind	.con
Argument	M	
Access Specification	M	
Index	S	
Domain Name	S	
Result(+)		S
Result(-)		S
Error Type		M

Argument
The argument shall convey the service specific parameters of the service request.

Access Specification
This parameter shall specify if the Index or the Domain Name is used to address the object.

Index
This parameter shall specify the logical address of the Domain.

Domain Name
This parameter shall specify the name of the Domain.

Result(+)
The Result(+) parameter shall indicate that the service was executed successfully.

Result(-)
The Result(-) parameter shall indicate that the service was not executed successfully.

Error Type
This parameter shall provide the reason for failure.

4.5.3.2 DownloadSegment

The DownloadSegment service shall be used to transfer one data segment into the server's Domain. The segment is transmitted in the service response.

Table 20. DownloadSegment

Parameter Name	.req .ind	.res .con
Argument	M	
Access Specification	M	
Index	S	
Domain Name	S	
Result(+)		S
Load Data		M
More Follows		M
Result(-)		S
Error Type		M

Argument
The argument shall convey the service specific parameters of the service request.

Access Specification
This parameter shall specify if the Index or the Domain Name is used to address the object.

Index
This parameter shall specify the logical address of the Domain.

Domain Name
This parameter shall specify the name of the Domain.

Result(+)
The Result(+) parameter shall indicate that the requested service was executed successfully.

Load Data
This parameter shall contain the data to be downloaded.

More Follows
Shall indicate whether or not any additional data remains to be transmitted.

Result(-)
The Result(-) parameter shall indicate that the service request failed.

Error Type
This parameter shall provide the reason for failure.

4.5.3.3 TerminateDownloadSequence

The TerminateDownloadSequence service shall be used to finish the loading of the Domain whose Index or Domain Name is included in the service request.

Table 21. TerminateDownloadSequence

Parameter Name	.req	.ind	.res	.con
Argument	M			
Access Specification	M			
Index	S			
Domain Name	S			
Final Result	M			
Result(+)			S	
Result(-)			S	
Error Type			M	

Argument
The argument shall convey the service specific parameters of the service request.

Access Specification
This parameter shall specify if the Index or the Domain Name is used to address the object.

Index
This parameter shall specify the logical address of the Domain.

Domain Name
This parameter shall specify the name of the Domain.

Final Result
This parameter shall inform the client whether or not the server successfully finished the Download.

Result(+)
The Result(+) parameter shall indicate that the requested service has succeeded.

Result(-)
The Result(-) parameter shall indicate that the service request failed.

Error Type
This parameter shall provide the reason for failure.

4.5.3.4 RequestDomainDownload

The RequestDomainDownload service shall be used by a server to request the client to perform a Download into the Domain whose Index or Domain Name is included in the service request.

The service response with Result(+) is not sent until the Download Sequence has been completely finished.

Table 22. RequestDomainDownload

Parameter Name	.req	.res	.ind	.con
Argument	M			
Access Specification	M			
Index	S			
Domain Name	S			
Additional Information	U			
Result(+)		S		
Result(-)		S		
Error Type		M		

Argument
The argument shall convey the service specific parameters of the service request.

Access Specification
This parameter shall specify if the Index or the Domain Name is used to address the object.

Index
This parameter shall specify the logical address of the Domain.

Domain Name
This parameter shall specify the name of the Domain.

Additional Information
This parameter shall contain additional information on the Domain Name. It may for example contain a file name. The correlation between Domain Name and file name is a local matter and is the responsibility of the application.

Result(+)
The Result(+) parameter shall indicate that the requested service was executed successfully.

Result(-)
The Result(-) parameter shall indicate that the service was not executed successfully.

Error Type
This parameter shall provide the reason for failure.

4.5.4 Upload Services

The Upload Services shall be used to transmit the data from the server's Domain to the client.

4.5.4.1 InitiateUploadSequence

The InitiateUploadSequence service shall be used to begin the Upload of the Domain whose Index or Domain Name is included in the service request.

Table 23. InitiateUploadSequence

Parameter Name	.req	.res	.ind	.con
Argument	M			
Access Specification	M			
Index	S			
Domain Name	S			
Result(+)		S		
Result(-)		S		
Error Type		M		

Argument
The argument shall convey the service specific parameters of the service request.

Access Specification
This parameter shall specify if the Index or the Domain Name is used to address the object.

Index
This parameter shall specify the logical address of the Domain.

Domain Name
This parameter shall specify the name of the Domain.

Result(+)
The Result(+) parameter shall indicate that the requested service was executed successfully.

Result(-)
The Result(-) parameter shall indicate that the service was not executed successfully.

Error Type
This parameter shall provide the reason for failure.

4.5.4.2 UploadSegment

The UploadSegment service shall be used to transfer one data segment of the server's Domain to the client.

Table 24. UploadSegment

Parameter Name	.req .ind	.res .con
Argument	M	
Access Specification	M	
Index	S	
Domain Name	S	
Result(+)		S
Load Data		M
More Follows		M
Result(-)		S
Error Type		M

Argument
The argument shall convey the service specific parameters of the service request.

Access Specification
This parameter shall specify if the Index or the Domain Name is used to address the object.

Index
This parameter shall specify the logical address of the Domain.

Domain Name
This parameter shall specify the name of the Domain.

Result(+)
The Result(+) parameter shall indicate that the requested service was executed successfully.

Load Data
This parameter shall contain the data requested from the server's Domain.

More Follows
This parameter shall indicate whether or not any additional data remains to be transmitted.

Result(-)
The Result(-) parameter shall indicate that the service was not executed successfully.

Error Type
This parameter shall provide the reason for failure.

4.5.4.3 TerminateUploadSequence

The TerminateUploadSequence service shall be used to finish the Upload Sequence.

Table 25. TerminateUploadSequence

Parameter Name	.req .ind	.res .con
Argument	M	
Access Specification	M	
Index	S	
Domain Name	S	
Result(+)		S
Result(-)		S
Error Type		M

Argument
The argument shall convey the service specific parameters of the service request.

Access Specification
This parameter shall specify if the Index or the Domain Name is used to address the object.

Index
This parameter shall specify the logical address of the Domain.

Domain Name
This parameter shall specify the name of the Domain.

Result(+)
The Result(+) parameter shall indicate that the requested service was executed successfully.

Result(-)
The Result(-) parameter shall indicate that the service was not executed successfully.

Error Type
This parameter shall provide the reason for failure.

4.5.4.4 RequestDomainUpload

The RequestDomainUpload service shall be used by a server to request that the contents of the specified Domain be uploaded to the client. The service response with Result(+) shall not be sent until the Upload Sequence has been completely finished.

Table 26. RequestDomainUpload

Parameter Name	.req .ind	.res .con
Argument	M	
Access Specification	M	
Index	S	
Domain Name	S	
Additional Information	U	
Result(+)		S
Result(-)		S
Error Type		M

Argument
The argument shall convey the service specific parameters of the service request.

Access Specification
This parameter shall specify if the Index or the Domain Name is used to address the object.

Index
This parameter shall specify the logical address of the Domain.

Domain Name
This parameter shall specify the name of the Domain.

Additional Information
This parameter shall contain additional information on the Domain Name. It may for example contain a file name. The correlation between Domain Name and file name is a local matter and is the responsibility of the application.

Result(+)
The Result(+) parameter shall indicate that the requested service was executed successfully.

Result(-)
The Result(-) parameter shall indicate that the service was not executed successfully.

Error Type
This parameter shall provide the reason for failure.

4.5.5 State Machine for Download

4.5.5.1 State Machine Description

EXISTENT
The Domain exists, but the content is not defined.

LOADING
Download on the Domain is in progress.

INCOMPLETE
Download failed, the content is incomplete.

COMPLETE
The Domain content has been transmitted, but the Domain is not yet released for use.

READY
The Domain is released for use.

IN-USE
The Domain is being used by a Program Invocation. Download is not allowed in this state.

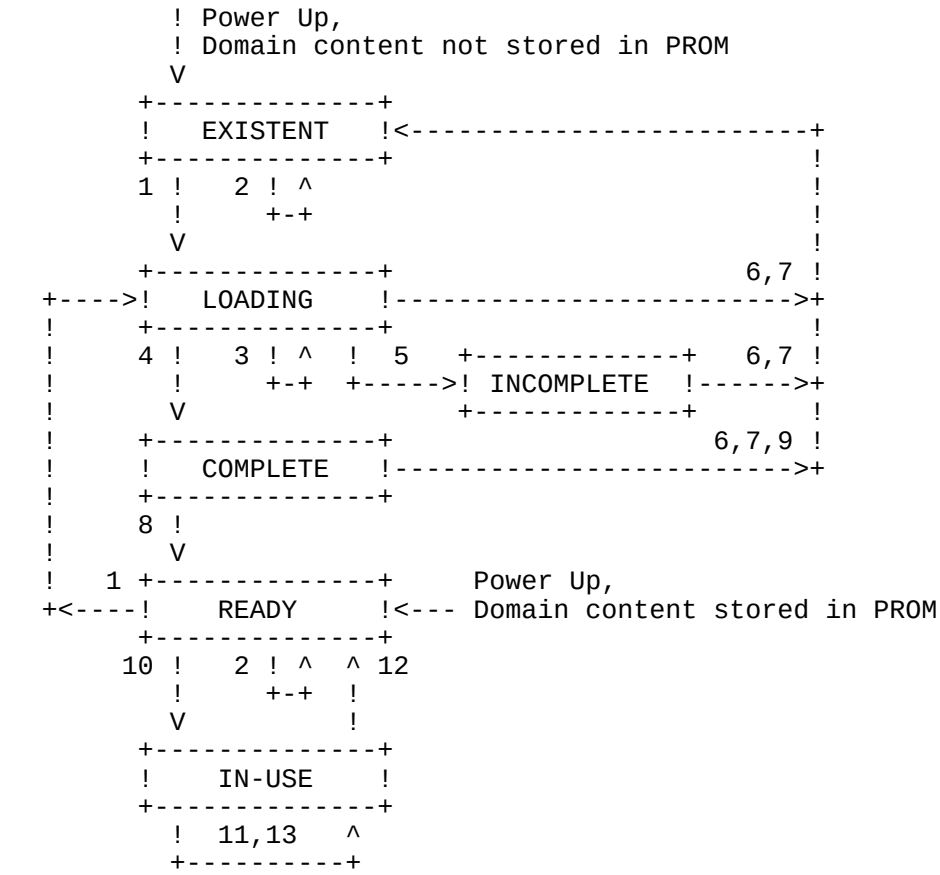


Figure 49. DomainDownload State Machine (Server)

Download and Upload shall not operate on the same Domain at the same time.

4.5.5.2 State Transitions

```
-------------------------------------------------------------------
    Event
        \Exit Condition
        =>  Action Taken
-------------------------------------------------------------------
 1  InitiateDownloadSequence.ind
        \state machine Upload in state NON-EXISTENT
        =>  .res(+)
 2  InitiateDownloadSequence.ind
        \state machine Upload not in state NON-EXISTENT
        =>  .res(-)  object constraint conflict
 3  DownloadSegment.con(+)
        \more follows = true
 4  DownloadSegment.con(+)
        \more follows = false
 5  DownloadSegment.con(-)
 6  TerminateDownloadSequence.req
        \final result = false
        \.con(+/-)
 7  Abort
 8  TerminateDownloadSequence.req
        \final result = true
        \.con(+)
 9  TerminateDownloadSequence.req
        \final result = true
        \.con(-)

See also State Transitions Program Invocation:

10  Counter increment (Start/Resume)
        =>  counter := 1
11  Counter increment (Start/Resume)
        =>  counter := counter +1
12  Counter decrement (Stop/Kill/End of Program/Program Stop)
        \counter = 1
        =>  counter := 0
13  Counter decrement (Stop/Kill/End of Program/Program Stop)
        \counter > 1
        =>  counter := counter -1
-------------------------------------------------------------------
```

4.5.6 State Machine for Upload

4.5.6.1 State Machine Description

NON-EXISTENT
The Domain exists, no Upload in progress.

UPLOADING
Upload on the Domain is in progress, Download is not allowed.

UPLOADED
The Domain content has been completely transmitted, but the TerminateUpload-Sequence service is still to be executed.

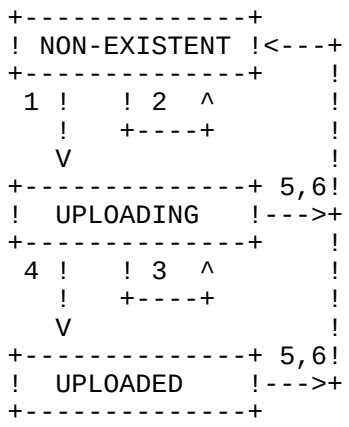


Figure 50. DomainUpload State Machine (Server)

Download and Upload shall not operate on the same Domain at the same time.

3.5.6.2 State Transitions

Event \ Exit Condition => Action Taken	
1	InitiateUploadSequence.ind \state machine Download in state EXISTENT or READY or IN-USE => .res(+)
2	InitiateUploadSequence.ind \state machine Download in state LOADING or INCOMPLETE or COMPLETE => .res(-) object constraint conflict
3	UploadSegment.ind \more follows = true => .res(+)
4	UploadSegment.ind \more follows = false => .res(+)
5	TerminateUploadSequence.res(+/-)
6	Abort.ind

4.5.7 EXAMPLES

4.5.7.1 EXAMPLE of a Download Sequence

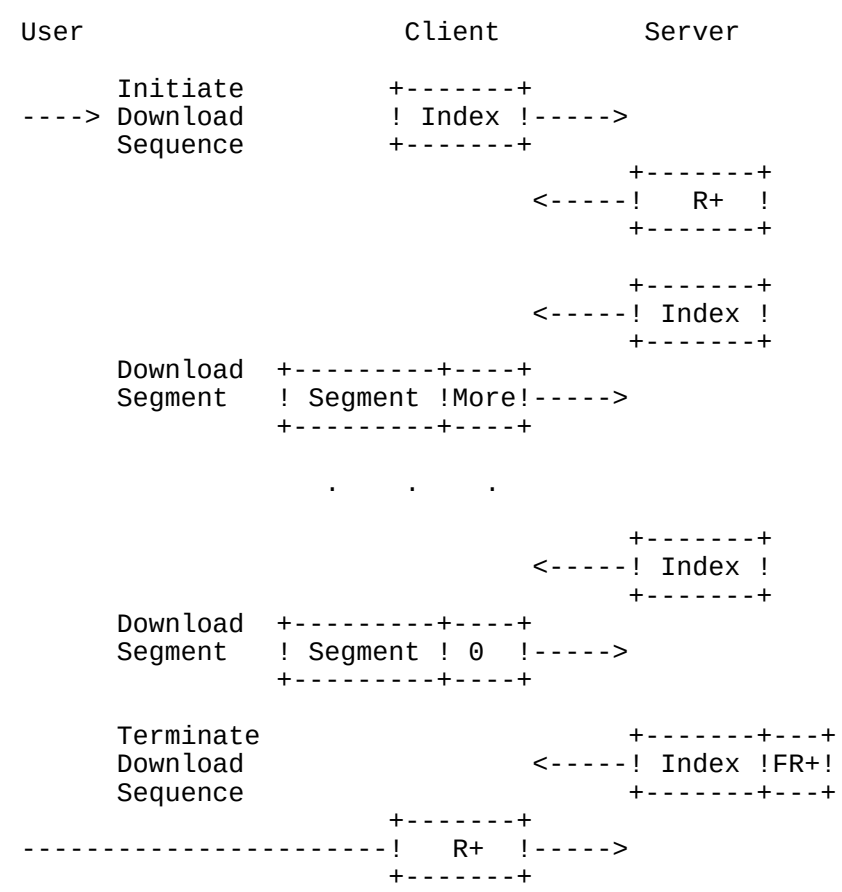


Figure 51. Example of a Download Sequence

4.5.7.2 EXAMPLE of an Upload Sequence

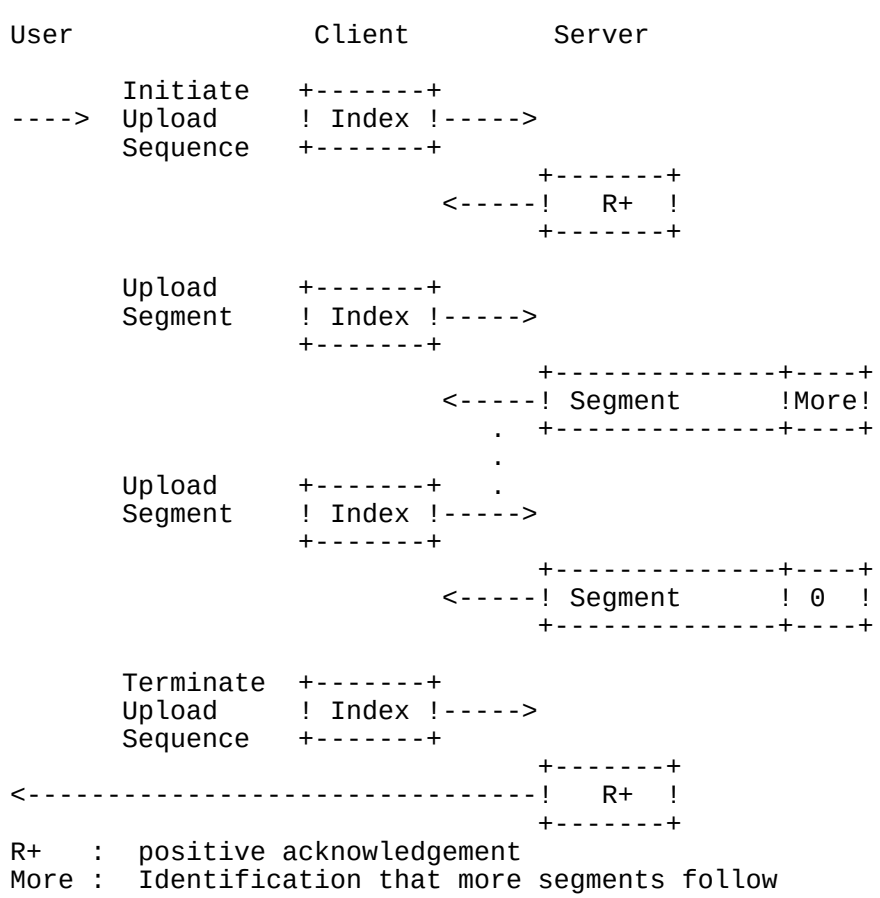


Figure 52. Example of an Upload Sequence

4.6 Program Invocation Management

Services on the object Program Invocation:

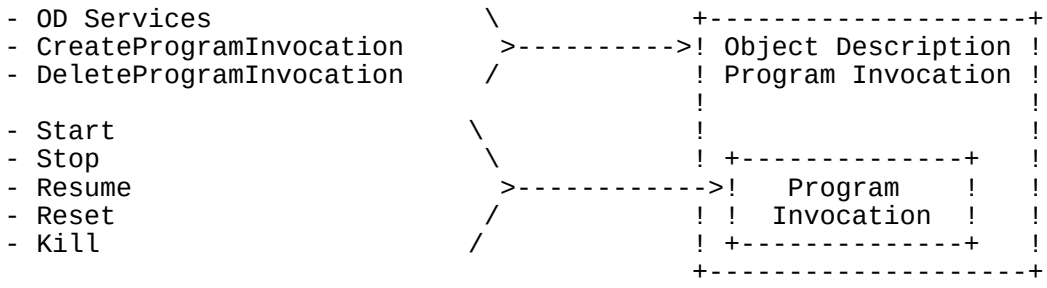


Figure 53. Program Invocation Services

4.6.1 Model Description

The Program Invocation model provides services to link domains to a program, to start this program, to stop and to delete it. More than one Program Invocation may be created in a device. A Program Invocation is defined by an entry in the DP-OD.

4.6.2 The Program Invocation (PI) Object

A Program Invocation is an object, in which domains with code and data are combined to an executable program. Program Invocations may be predefined or may be created online. When the Object Dictionary (OD) is freshly loaded, all Program Invocation Objects are deleted.

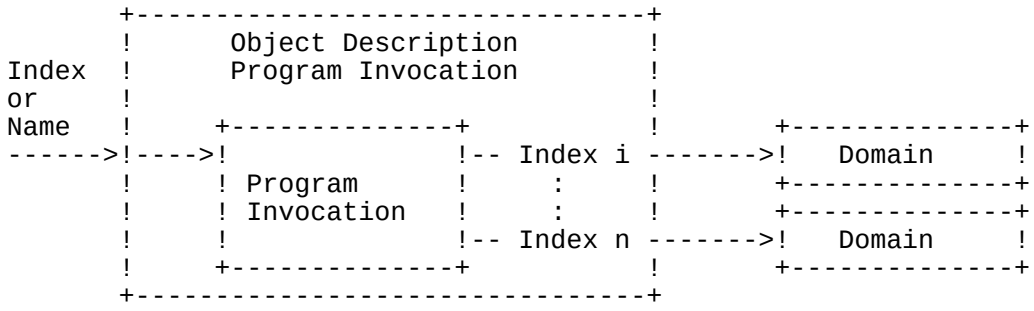


Figure 54. Overview of the Program Invocation

4.6.2.1 Attributes

Object: Program Invocation
Key Attribute: Index
Key Attribute: PI name
Attribute: Number Of Domains
Attribute: Password
Attribute: Access Groups
Attribute: Access Rights
Attribute: Deletable
Attribute: Reusable
Attribute: PI State
Attribute: List of Domain Index
Attribute: Index
Attribute: Extension

Index
Logical address of the entry in the OD.

PI name
Name of the object. The existence and the length is defined in the OD object description.

Number Of Domains
Number of the domain indices entered in this Program Invocation. The number is limited by the maximum PDU length of the Create Program Invocation service.

Password
This attribute contains the password for the Access Rights.

Access Groups
This Attribute relates the object to specific Access Groups. The object belongs to the group when the corresponding bit is set.

Table 27. Access Groups for Program Invocation

Bit	Meaning
7	Access Group 1
6	Access Group 2
5	Access Group 3
4	Access Group 4
3	Access Group 5
2	Access Group 6
1	Access Group 7
0	Access Group 8

Access Rights

This attribute contains information about the Access Rights. The specific Access Right exists when the corresponding bit is set.

Table 28. Access Rights for PI

Bit	Name	Meaning
7	S	Right to Start the PI for the registered Password (Start, Resume, Reset)
6	H	Right to Stop the PI for the registered Password (Stop)
5	D	Right to Delete the PI for the registered Password (Kill, Delete Program Invocation)
3	Sg	Right to Start for Access Groups
2	Hg	Right to Stop for Access Groups
1	Dg	Right to Delete for Access Groups
15	Sa	Right to Start for all Communication Partners
14	Ha	Right to Stop for all Communication Partners
13	Da	Right to Delete for all Communication Partners

Deletable

This attribute indicates if the Program Invocation Object may be deleted using the Delete Program Invocation service.

true <=> deletable
false <=> not deletable.

Reusable

This attribute indicates if the Program Invocation transits after execution to the state IDLE or to the state UNRUNNABLE.

true <=> transits to the state IDLE
false <=> transits to the state UNRUNNABLE.

PI State

This attribute describes the state of the program.

1 <=> UNRUNNABLE
2 <=> IDLE
3 <=> RUNNING
4 <=> STOPPED
5 <=> STARTING
6 <=> STOPPING
7 <=> RESUMING
8 <=> RESETTING

List of Domain Index

This attribute contains the indices of the domains, which are combined to this Program Invocation. The ordering of the entries of the domain indices in the Program Invocation corresponds to the ordering in the Create Program Invocation service. The first domain in the list of the indices shall contain an executable program.

Extension

This attribute contains profile specific information.

4.6.2.2 Object Description of the OD on Transmission

Index	Object code	Number of Domains	Password	Access Groups	Access Rights
130	PI	7	17		S H D

Deletable	Reusable	PI State	Domain Index (1)	Domain Index (2)
true	true		34	55

Domain Index (3)	Domain Index (4)	Domain Index (5)	Domain Index (6)
56	66	67	68

Domain Index (7)	PI Name	Extension
77	P. Thiessmeier	

Figure 55. Object Description of Program Invocation in the DP-OD (Example)

Object Code
Tag for the object Program Invocation, is transmitted in addition to the object attributes for the GetOD service and the PutOD service.

4.6.3 Program Invocation Services

4.6.3.1 CreateProgramInvocation

With this service, domains (e.g programs, data regions), that have been defined in the OD, are combined to a program and are defined online in the DP-OD. This program is addressed by a logical address or a name. The Deletable attribute of a Program Invocation, that is created by the Create Program Invocation service, is set to true. The attribute Number of Domains in the object description is created automatically by the server.

The first domain of the list of domains shall contain an executable program.

The Program Invocation is only created, if the rights for the use of the domains exist. Otherwise a Result(-) is responded.

If the same Program Invocation with the same Access Rights already exists, no new object is created. In this case, the logical address (Index) of the existing object is given to the client.

Table 29. Create Program Invocation

Parameter Name	.req .ind	.res .con
Argument	M	
Password	M	
Access Groups	M	
Access Rights	M	
Reusable	M	
List of Domains	M	
Domain Index	S	
Domain Name	S	
PI Name	U	
Extension	U	
Result(+)		S
Index		M
Result(-)		S
Error Type		M

Argument

The argument contains the service specific parameters of the service request. If Access Rights are not supported (Access Protection Supported = false), then the parameters Password, Access Groups and Access Rights are not significant.

Password
This parameter states to which value the attribute Password shall be set.

Access Groups
This parameter states to which value the attribute Access Groups shall be set.

Access Rights
This parameter states to which value the attribute Access Rights shall be set.

Reusable
This parameter states to which value the attribute reusable shall be set.

List of Domains
This parameter contains the addresses of the domains, which shall be combined in this Program Invocation to a program. Each address may be either a logical address or a name.

Domain index
Index of a domain.

Domain name
Name of a domain.

PI Name
This parameter states, to which value the attribute name of the Program Invocation Object shall be set.

Extension
This parameter contains the Extension attribute.

Result(+)
The parameter shall indicate, that the service was completed successfully.

Index:
The index that is associated with the created Program Invocation.

Result(-):
The Result(-) shall indicate that the service request was not completed successfully.

Error Type:
This parameter contains information about the reason, why the service was not completed successfully.

4.6.3.2 DeleteProgramInvocation

With the Delete Program Invocation service programs that have been defined in the DP-OD are deleted. Program Invocation Objects may only be deleted if their attribute Deletable has the value true.

Table 30. Delete Program Invocation

Parameter Name	.req	.res	.ind	.con
Argument	M			
Access Specification	M			
Index	S			
PI Name	S			
Result(+)		S		
Result(-)		S		
Error Type		M		

Argument
The argument contains the service specific parameters of the service request.

Access Specification
This parameter states if the index or the name is used for addressing.

Index
This parameter is the Logical Address of the Program Invocation in the OD to which this service is related.

PI Name
This parameter is the name of the Program Invocation.

Result(+)
The Result(+) parameter shall indicate, that the service was completed successfully.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

4.6.3.3 Start

A program is started with this service. It runs from the beginning. The counters of the used domains are incremented. Before that all corresponding domains are checked if they are in the READY or in the IN USE state.

Table 31. Start

Parameter Name	.req .ind	.res .con
Argument	M	
Access Specification	M	
Index	S	
PI Name	S	
Execution Argument	U	
Result(+)		S
Result(-)		S
Error Type		M
PI State		M

Argument
The argument contains the service specific parameters of the service request.

Access Specification
This parameter states if the index or the name is used for addressing.

Index
This parameter is the logical address of the Program Invocation in the OD to which this service is related.

PI Name
This parameter is the name of the Program Invocation.

Execution Argument
This optional parameter of type octet string contains data that is given to the Program Invocation for the start.

Result(+)
The Result(+) parameter shall indicate, that the service was completed successfully.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

PI State
This parameter indicates the state of the Program Invocation.

4.6.3.4 Stop

A running program is stopped. It is not set to the beginning. The counters of the used domains are decremented.

Table 32. Stop

Parameter Name	.req .ind	.res .con
Argument	M	
Access Specification	M	
Index	S	
PI Name	S	
Result(+)		S
Result(-)		S
Error Type		M
PI State		M

Argument
The argument contains the service specific parameters of the service request.

Access Specification
This parameter states if the index or the name is used for addressing.

Index
This parameter is the logical address of the Program Invocation in the OD to which this service is related.

PI Name
This parameter is the name of the Program Invocation.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

PI State
This parameter indicates the state of the Program Invocation.

4.6.3.5 Resume

A stopped program is set to the state RUNNING. It is not reset. The counters of the used domains are incremented. Before that all corresponding domains are checked if they are in the READY or in the IN USE state.

Table 33. Resume

Parameter Name	.req .ind	.res .con
Argument	M	
Access Specification	M	
Index	S	
PI Name	S	
Execution Argument	U	
Result(+)		S
Result(-)		S
Error Type		M
PI State		M

Argument
The argument contains the service specific parameters of the service request.

Access Specification
This parameter states if the index or the name is used for addressing.

Index
This parameter is the logical address of the Program Invocation in the OD to which this service is related.

PI Name
This parameter is the name of the Program Invocation.

Execution Argument
This optional parameter of type octet string contains data that is given to the Program Invocation for the start.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

PI State
This parameter indicates the state of the Program Invocation.

4.6.3.6 Reset

A stopped program having the attribute Reusable = true is set to the beginning. The Program Invocation transits to the state IDLE. If the attribute Reusable is equal to false the Program Invocation Object transits to the state UNRUNNABLE.

Table 34. Reset

Parameter Name	req	ind	res	con
Argument	M			
Access Specification	M			
Index	S			
PI Name	S			
Result(+)			S	
Result(-)			S	
Error Type			M	
PI State			M	

Argument
The argument contains the service specific parameters of the service request.

Access Specification
This parameter states if the index or the name is used for addressing.

Index
This parameter is the logical address of the Program Invocation in the OD to which this service is related.

PI Name
This parameter is the name of the Program Invocation.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

PI State
This parameter indicates the state of the Program Invocation.

4.6.3.7 Kill

A Program Invocation is set to the state UNRUNNABLE. This is independent of the current state. If the state of the Program Invocation was RUNNING or STOPPING, the counters of the used domains are decremented.

Table 35. Kill

Parameter Name	.req .ind	.res .con
Argument	M	
Access Specification	M	
Index	S	
PI Name	S	
Result(+)		S
Result(-)		S
Error Type		M

Argument
The argument contains the service specific parameters of the service request.

Access Specification
This parameter states if the index or the name is used for addressing.

Index
This parameter is the logical address of the Program Invocation in the OD to which this service is related.

PI Name
This parameter is the name of the Program Invocation.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

4.6.4 State Machine

4.6.4.1 State Machine Description

NON-EXISTENT
The Program Invocation is after a power-up, if not otherwise defined, or after deletion, non-existent.

IDLE
A Program Invocation transits to the state IDLE by entering it in the DP-OD. After a successful Reset service and after the end of the program the Program Invocation transits also to the state IDLE. Program Invocations may be predefined. These Program Invocations transit after power-up to the state IDLE.

STARTING
Intermediate state for the preparation of the program start.

RUNNING
The program is running in this state. The domains that are defined in this Program Invocation are now in the state IN USE.

STOPPING
Intermediate state for the preparation of the stop of the program.

STOPPED
The program is stopped in this state.

RESUMING
Intermediate state for the preparation of the resumption of the program.

RESETTING
Intermediate state in which the program is set to a well defined initial state.

UNRUNNABLE
In this state the program is stopped but it may not be started again. It may only be deleted.

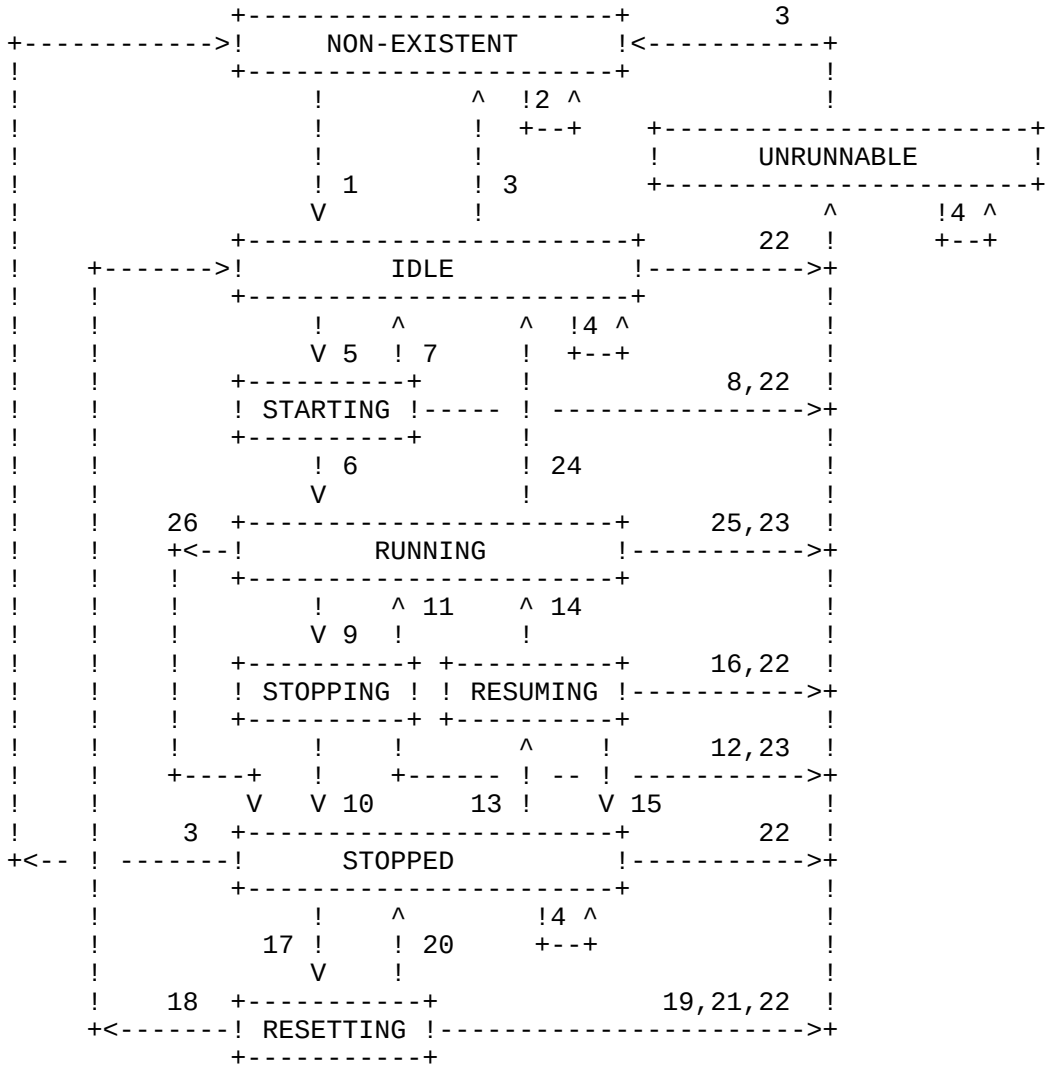


Figure 56. State Diagram

4.6.4.2 State Transitions

	Event \ Exit Condition => Action Taken
1	CreateProgramInvocation.ind \Domains available AND Access to Domains allowed => .res(+)
2	CreateProgramInvocation.ind \Domains not available OR Access to Domains not allowed => .res(-)
3	DeleteProgramInvocation.ind => .res(+)
4	DeleteProgramInvocation.ind => .res(-)
5	Start.ind

```
-----------------------------------------------------------------
    Event
        \Exit Condition
        =>  Action Taken
-----------------------------------------------------------------

6  Start successfully executed
         \Domains in State READY or IN-USE
         =>  Counter increment
             Start.res(+)
 7  Start failed, non-destructive
         (e.g. Domains not in State READY or IN-USE)
         =>  Start.res(-)
 8  Start failed, destructive
         =>  Start.res(-)
 9  Stop.ind
10  Stop successfully executed
         =>  Counter decrement
             Stop.res(+)
11  Stop failed, non-destructive
         =>  Stop.res(-)
12  Stop failed, destructive
         =>  Counter decrement
             Stop.res(-)
13  Resume.ind
14  Resume successfully executed
         \Domains in State READY or IN-USE
         =>  Counter increment
             Resume.res(+)
15  Resume failed, non-destructive
         (e.g. Domains not in State READY or IN-USE)
         =>  Resume.res(-)
16  Resume failed, destructive
         =>  Resume.res(-)
17  Reset.ind
18  Reset successfully executed, Reusable = true
         =>  Reset.res(+)
19  Reset successfully executed, Reusable = false
         =>  Reset.res(+)
20  Reset failed, non-destructive
         =>  Reset.res(-)
21  Reset failed, destructive
         =>  Reset.res(-)
22  Kill.ind
         =>  .res(+)
23  Kill.ind
         =>  Counter decrement
             .res(+)
24  End of Program, Reusable = true
         =>  Counter decrement
25  End of Program, Reusable = false
         =>  Counter decrement
26  Program stop
         =>  Counter decrement
```

Counter increment, Counter decrement: see StateMachineDomain Download.

4.7 Variable Access

4.7.1 Model Description

The Variable Access Model provides services to read and to write variables and to define and to delete dynamically new Variable List Objects.

The following services and objects are defined:

Table 36. Variable Access Services and Objects

Service	Confirmed/ Unconfirmed	Objects
Read	Confirmed	Simple Variable, Array, Record, Variable List
Write	Confirmed	
ReadWithType	Confirmed	
WriteWithType	Confirmed	
InformationReport	Unconfirmed	
InformationReport-WithType	Unconfirmed	
PhysRead	Confirmed	Physical Access Object
PhysWrite	Confirmed	
DefineVariableList	Confirmed	Object Description Variable List,
DeleteVariableList	Confirmed	Variable List
---	---	Data Type
---	---	Date Type Structure Description

4.7.2 Variable Access Objects

Objects entered in the S-OD are not deletable.

4.7.2.1 The Physical Access Object

The Physical Access Object describes the access to an actual octet string.

Such an octet string may neither be defined nor be deleted.

No explicit object description of the Physical Access Object is stored. The address and semantics are known to the user.

The access rights are controlled through the Service Context Agreement, i.e. the Context Agreement specifies if the physical access is allowed on this Communication Relationship.

Services onto the Physical Access Object:

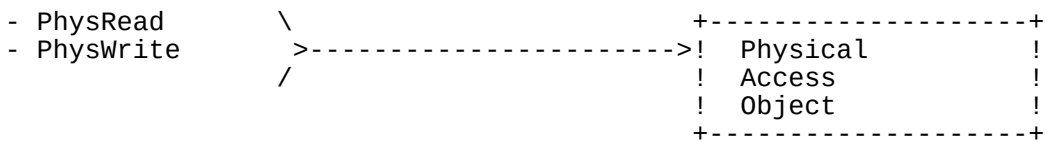


Figure 57. Physical Access Object Services

Attributes
Object: Physical Access Object
Key Attribute: Local Address
Attribute: Length

Local Address
This object is accessed knowing its physical address and its structure, by-passing the Object Dictionary of the related device. The data type octet string is used implicitly.

Length
Statement of the length in octets for this object during read and write operations.

4.7.2.2 The Simple Variable Object

The Simple Variable Object represents a single, simple variable which is characterized by a defined Data Type. The Object Description of the Variable Access Object Simple Variable is stored statically in the Object Dictionary (S-OD). The mapping of a PROFIBUS Simple Variable to a real Simple Variable, which exists in the application system, is defined by the object description of the Simple Variable Object. Only one single Data Type of the set of configured Data Types is allowed in the Object Description of the Simple Variable Object.

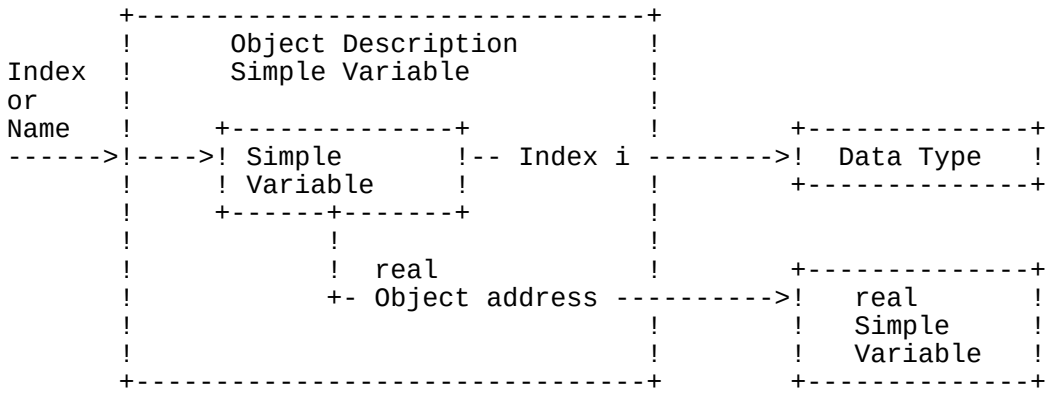


Figure 58. Overview of Simple Variable

Services on the Object Simple Variable:

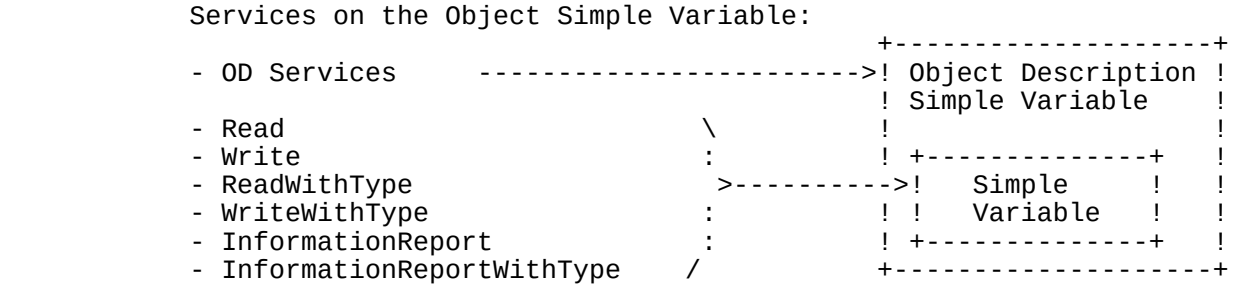


Figure 59. Simple Variable Servioces

Attributes

Object: Simple Variable
  Key Attribute: Index
  Key Attribute: Variable Name
  Attribute: Data Type Index
  Attribute: Length
  Attribute: Password
  Attribute: Access Groups
  Attribute: Access Rights
  Attribute: Local Address
  Attribute: Extension

Index
Logical Address of the object.

Variable Name
The name of the object. Its existence and its length is defined in the OD Object Description.

Data Type Index
Logical address of the corresponding Data Type in the Static Type Dictionary (ST-OD).

Length
Length in octets of the data.

Password
This attribute contains the password for the Access Rights.

Access Groups
This attribute relates the object to specific Access Groups. The object is a group member when the corresponding bit is set.

Table 37. Access Groups for Simple Variable

Bit	Meaning
7	Access Group 1
6	Access Group 2
5	Access Group 3
4	Access Group 4
3	Access Group 5
2	Access Group 6
1	Access Group 7
0	Access Group 8

Access Rights
This attribute contains information about the Access Rights. The specific Access Right exists when the corresponding bit is set.

Table 38. Access Rights for Simple Variable

Bit	Name	Meaning
7	R	Right to Read for the registered Password
6	W	Right to Write for the registered Password
3	Rg	Right to Read for Access Groups
2	Wg	Right to Write for Access Groups
15	Ra	Right to Read for all Communication Partners
14	Wa	Right to Write for all Communication Partners

Local Address:
The Local Address is a system specific address of the real object. It serves the internal addressing of the object. If no mapping of this kind is performed, this attribute shall have the value FFFFFFFF hex.

Extension:
This attribute contains profile specific information.

Object Description of the OD on Transmission

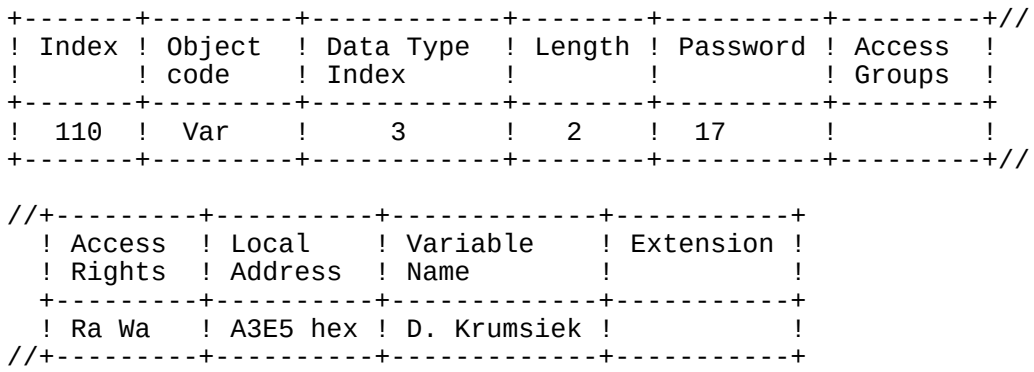


Figure 60. Object Description of Simple Variable in the S-OD (Example)

Object Code
Tag for the Simple Variable Object, which is transmitted in addition to the object attributes in the GetOD service and the PutOD service.

4.7.2.3 The Array Object

The Array Object consists of a collection of Simple Variables of the same Data Type.

The Object Description of the Variable Access Object Array is stored statically in the Object Dictionary (S-OD). The mapping of a PROFIBUS Array to a real Array, which exists in the application system, is described by the object description of this object. The object description states the relation of the object to the Data Type. Only one single Data Type of the set of configured Data Types is allowed for all Array Elements.

The Array Object may be accessed completely or element by element. If the Object shall be accessed completely the index shall be used for the service. If an element of the object shall be accessed then the subindex shall be used together with the index for the service. Subindex 1 accesses the first element of the object.

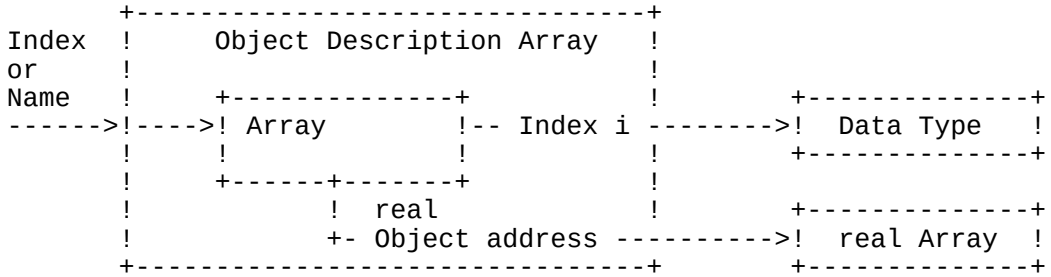


Figure 61. Overview of Array

Services on the Object Array:

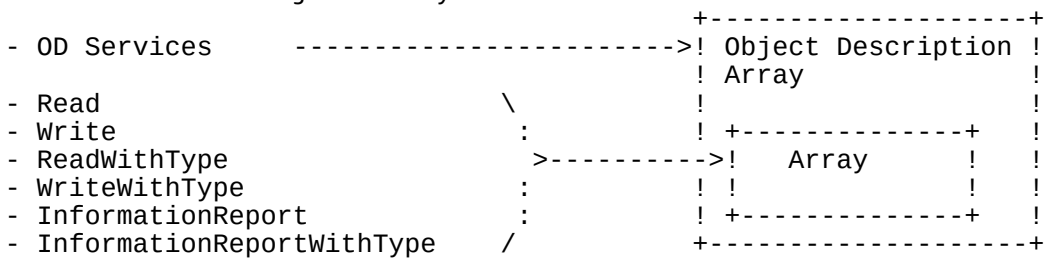


Figure 62. Array Services

Attributes

Object: Array
Key Attribute: Index
Key Attribute: Variable Name
Attribute: Data Type Index
Attribute: Length
Attribute: Number Of Elements
Attribute: Password
Attribute: Access Groups
Attribute: Access Rights
Attribute: Local Address
Attribute: Extension

Index
Logical address of the object.

Variable Name
The name of the object. Its existence and its length is defined in the OD Object Description.

Data Type Index
Logical address of the corresponding Data Type in the Static Type Dictionary.

Length
Length in octets of one Array Element.

Number Of Elements
States how many elements are contained in the Array.

Password
This attribute contains the password for the Access Rights.

Access Groups

This attribute relates the object to the specific Access Groups. The object is a group member when the corresponding bit is set.

Access Rights
This attribute contains information about the Access Rights. The specific Access Right exists when the corresponding bit is set.

Table 39. Access Groups for Array

Bit	Meaning
7	Access Group 1
6	Access Group 2
5	Access Group 3
4	Access Group 4
3	Access Group 5
2	Access Group 6
1	Access Group 7
0	Access Group 8

Table 40. Access Rights for Arrays

Bit	Name	Meaning
7	R	Right to Read for the registered Password
6	W	Right to Write for the registered Password
3	Rg	Right to Read for Access Groups
2	Wg	Right to Write for Access Groups
15	Ra	Right to Read for all Communication Partners
14	Wa	Right to Write for all Communication Partners

Local Address
The Local Address is a system specific address of the real object. It serves the internal addressing of the object. If no mapping of this kind is performed this attribute shall have the value FFFFFFFF hex.

Extension
This attribute contains profile specific information.

Object Description of the OD on Transmission

Index	Object code	Data Type Index	Length	Number Of Elements	Password
121	Array	5	1	6	34

Access Groups	Access Rights	Local Address	Variable Name	Extension
	R W	1234 hex	M. Braun	

Figure 63. Object Description of Array in the S-OD (Example)

Object Code
Tag for the Array Object, which is transmitted in addition to the object attributes in the GetOD service and the PutOD service.

4.7.2.4 The Record Object

The Record Object consists of a collection of Simple Variables of different Data Types.

The Object Description of the Variable Access Object Record is stored dynamically in the Object Description (S-OD).

The mapping of a PROFIBUS Record to a real Record, which exists in the application system, is described by the object description of this object. The object description states the relation of the object to a configured Data Type Structure Description.

The Record Object may be accessed completely or element-wise. If the Object shall be accessed completely the index shall be used for the service. If an element of the object shall be accessed then the subindex shall be used together with the index for the service. Subindex 1 accesses the first element of the object.

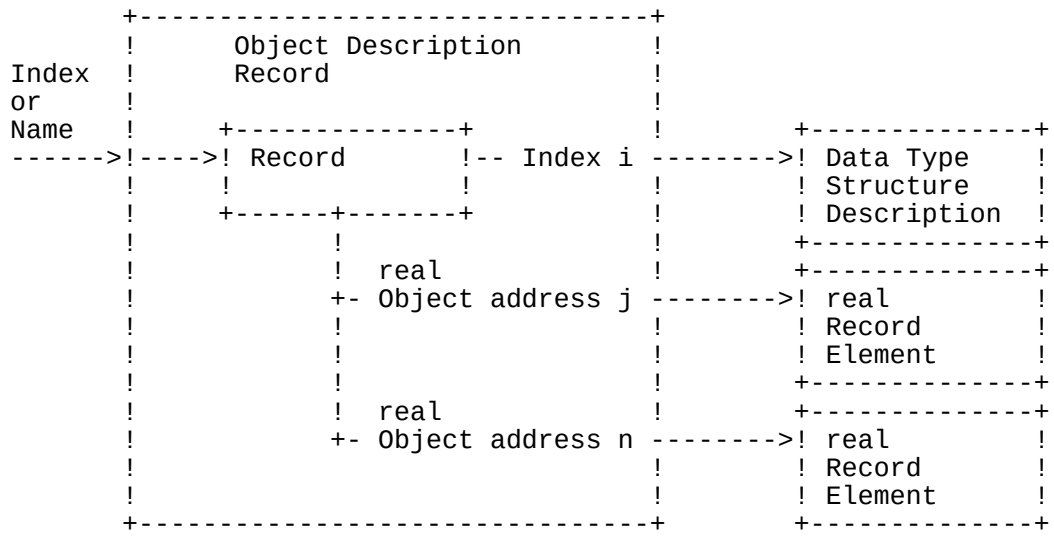


Figure 64. Overview of Record

Services on the Object Record:

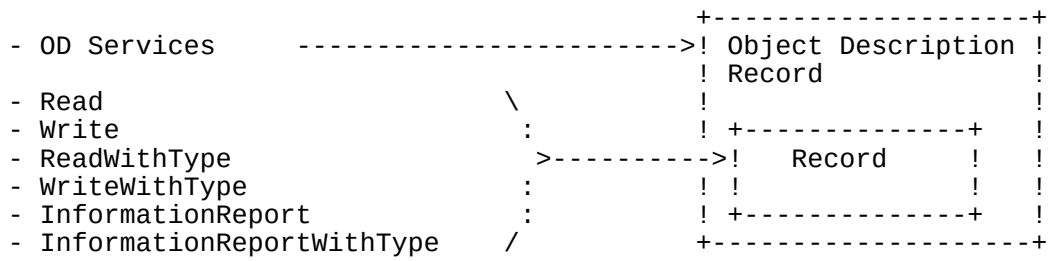


Figure 65. Record Services

Attributes

```
Object:  Record
  Key Attribute:  Index
  Key Attribute:  Variable Name
  Attribute:  Data Type Index
  Attribute:  Password
  Attribute:  Access Groups
  Attribute:  Access Rights
  Attribute:  Extension
  Attribute:  List of Local Address
    Attribute:  Local Address
```

Index
Logical Address of the object.

Variable Name
The name of the object. Its existence and its length is defined in the OD Object Description.

Data Type Index
Logical Address of the related Data Type Structure Description in the Static Type Dictionary (ST-OD).

Password
This attribute contains the password for the Access Rights.

Access Groups
This attribute relates the object to specific Access Groups. The object is a group member when the corresponding bit is set.

Table 41. Access Groups for Record

Bit	Meaning
7	Access Group 1
6	Access Group 2
5	Access Group 3
4	Access Group 4
3	Access Group 5
2	Access Group 6
1	Access Group 7
0	Access Group 8

Access Rights
This attribute contains information about the Access Rights. The specific Access Right exists when the corresponding bit is set.

Table 42. Access Rights for Record

Bit	Name	Meaning
7	R	Right to Read for the registered Password
6	W	Right to Write for the registered Password
3	Rg	Right to Read for Access Groups
2	Wg	Right to Write for Access Groups
15	Ra	Right to Read for all Communication Partners
14	Wa	Right to Write for all Communication Partners

Extension
This attribute contains profile specific information.

List of Local Address
This attribute contains the internal addresses of the Record Elements. These addresses are system specific addresses of the real objects. They serve the internal addressing of the objects. If the real objects are stored sequentially without gaps, then only the Local Address (1) is given as start address. If no mapping of this kind is performed, this attribute shall be set to the value FFFFFFFF hex.

Object Description of the OD on Transmission

Index	Object code	Data Type Index	Password	Access Groups	Access Rights
111	Record	37	34		R

Variable Name	Extension	Local Address (1)	Local Address (2)
R. Breithaupt		43A7 hex	62AB hex

Local Address (3)	Local Address (4)	Local Address (5)
3942 hex	2324 hex	AC56 hex

Figure 66. Object Description of Record in the S-OD (Example)

Object Code
Tag for the Record Object, which is transmitted in addition to the object attributes in the GetOD service and the PutOD service.

4.7.2.5 The Variable List Object

The Variable List Object consists of a collection of Variable Access Objects.

The Object Description of the Variable Access Object Variable List is stored dynamically in the Object Dictionary (DV-OD). This Object Description contains an index list of Array, Record and Simple Variable Objects from the S-OD.

A Variable List may be created and deleted with the DefineVariableList and DeleteVariableList services.

The Access Rights shall be declared for the DefineVariableList service. The Access Rights to the single objects are checked on creation (DefineVariableList) of a Variable List. The Variable List Object is only created if the requested Access Rights do not exceed the Access Rights to the single objects.

If the same Variable List exists with the requested Access Rights, no new object is created but the Logical Address of the existing object is given to the client. In the other case the Variable List is created with the requested Access Rights.

Only the authorized client with the proper Access Rights may delete a Variable List Object (DeleteVariableList).

If the Object Dictionary is freshly loaded all Variable List Objects are deleted.

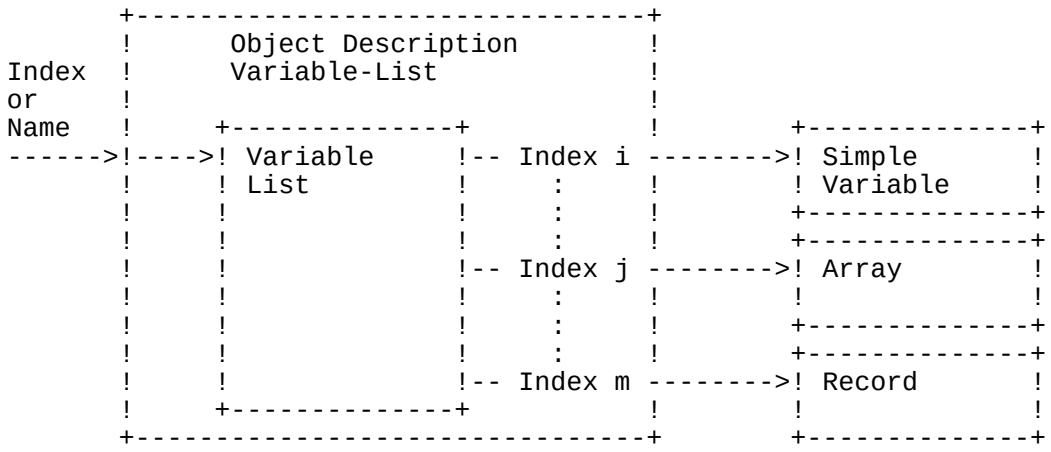


Figure 67. Overview of Variable List

Services on the Object Variable List:

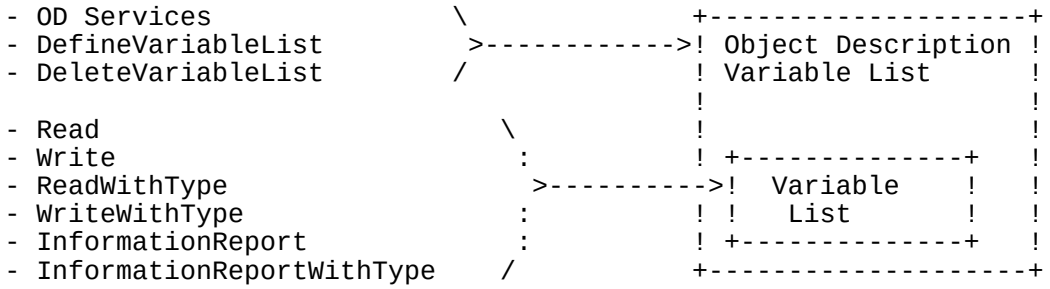


Figure 68. Variable List Services

Attributes

Object: Variable List
- Key Attribute: Index
- Key Attribute: Variable List Name
- Attribute: Number Of Elements
- Attribute: Password
- Attribute: Access Groups
- Attribute: Access Rights
- Attribute: Deletable
- Attribute: List of Element Index
  - Attribute: Index
- Attribute: Extension

Index
Logical Address of the object.

Variable List Name
The name of the object. Its existence and its length is defined in the OD Object Description.

Number Of Elements
This parameter states the number of Variable Access Objects of which the Variable List consists.

Password
This parameter contains the password for the Access Rights.

Access Groups
This attribute relates the object to specific Access Groups. The object is a group member when the corresponding bit is set.

Table 43. Access Groups for Variable List

Bit	Meaning
7	Access Group 1
6	Access Group 2
5	Access Group 3
4	Access Group 4
3	Access Group 5
2	Access Group 6
1	Access Group 7
0	Access Group 8

Access Rights
This attribute contains information about the Access Rights. The specific Access Right exists when the corresponding bit is set.

Table 44. Access Rights for Variable List

Bit	Name	Meaning
7	R	Right to Read for the registered Password
6	W	Right to Write for the registered Password
5	D	Right to Delete for the registered Password
3	Rg	Right to Read for Access Groups
2	Wg	Right to Write for Access Groups
1	Dg	Right to Delete for Access Groups
15	Ra	Right to Read for all Communication Partners
14	Wa	Right to Write for all Communication Partners
13	Da	Right to Delete for all Communication Partners

Deletable
This attribute indicates if the Variable List Object may be deleted (true) or not (false) using the DeleteVariableList service.

List of Element Index
This attribute contains the indices of the Variable Access Objects which are combined to this Variable List.

Extension
This attribute contains profile specific information.

Object Description of the OD on Transmission

Index	Object code	Number Of Elements	Password	Access Groups
220	Var-List	8	34	

Access Rights	Deletable	Element Index (1)	Element Index (2)	Element Index (3)
R W D	true	142	127	158

Element Index (4)	Element Index (5)	Element Index (6)	Element Index (7)	Element Index (8)
172	108	196	134	188

Variable List Name	Extension
Z. Szabo	

Figure 69. Object Description of Variable List in the DV-OD (Example)

Object Code
Tag for the Variable List Object, which is transmitted in addition to the object attributes for the GetOD service and the PutOD service.

4.7.2.6 The Data Type Object

A Data Type characterises a set of values and a set of operations that may be applied to these values. The Data Type defines the syntax, the range of the variables and their presentation inside the communication system. The Data Type of an object is indicated by a corresponding attribute in the Object Description. Type definitions may not be made remotely for reasons of efficiency. They have a fixed configuration in the Static Type Dictionary (ST-OD).

The PROFIBUS Specification defines free configurable Data Types and Standard Data Types. They are stored beginning with Index 1 in the Static Type Dictionary (ST-OD). Standard Data Types which are not used are marked as not configured in the ST-OD (see also data type object presentation in the OD).

Services on the Data Type Object:
- none

Attributes

Object: Data Type
  Key Attribute: Index
  Attribute: Description
    Attribute: Symbol Length
    Attribute: Symbol

Index
Logical Address of the object. (The semantic of the indices is fixed).

Description
This attribute contains a textual description of the Data Type. It consists of the Symbol Length and the Symbol.

Symbol Length
This attribute contains the length of the symbol. The value 0 means that no symbol is used.

Symbol
This attribute contains a visible string of 0 to 32 octets as a symbolic description.

Structure of the OD on Transmission

The entry for a Data Type in the ST-OD has the structure below.

Index	Object code	Description	
		Symbol Length	Symbol
01	Data Type	7	Boolean

Figure 70. Object Description of Data Type in the ST-OD

Object Code
Tag for the Data Type Object, which is transmitted in addition to the object attributes for the GetOD and the PutOD service.

Null Object <=> not configured.

EXAMPLE:

Index	Object code	Description	
01	Data Type	Boolean	Boolean
02	Null Object		Integer8, not configured
03	Data Type	Integer16	Integer16
...			
76	Data Type	xy special	Application specific

Figure 71. Object Description of Data Types in the ST-OD (Example)

4.7.2.7 The Data Type Structure Description Object

The Data Type Structure Description Object characterizes the size and the structure of records (see also Record Object).

The Object Description of the Data Type Structure Description Object has a fixed configuration in the Static Type Dictionary. It contains a list of elements, consisting of Data Type and Length of the corresponding record element. As Data Types for the single elements only Data Types in the set of configured Data Types of the Static Type Dictionary are allowed.

Services on the Data Type Structure Description Object:

- none

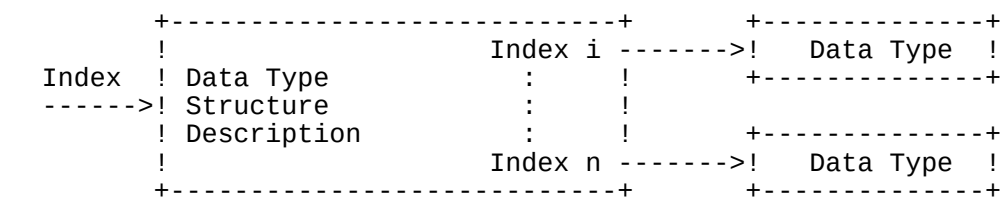


Figure 72. Overview of Data Type Structure Description

Attributes

Object: Data Type Structure Description
  Key Attribute: Index
  Attribute: Number Of Elements
  Attribute: List of Element
    Attribute: Data Type Index
    Attribute: Length

Index
Logical Address of the object.

Number Of Elements
Number of elements of the structured Data Type.

List of Element
This attribute contains a list of elements, consisting of Data Type and Length of the corresponding record element.

Data Type Index
Logical Address of the Data Type of element n.

Length
Length of the element n in octets.

EXAMPLE:

Structure of the OD on Transmission

Index	Object code	Number Of Elements	Data Type Index (1)	Length (1)
39	Ds	5	7	4

Data Type Index (2)	Length (2)	Data Type Index (3)	Length (3)
9	12	8	4

Data Type Index (4)	Length (4)	Data Type Index (5)	Length (5)
9	8	8	4

Figure 73. Object Description of Data Type Structure Description in the ST-OD (Example)

Object Code
Tag for the Data Type Structure Description which is transmitted in addition to the object attributes for the GetOD and the PutOD service.

4.7.3 Variable Access Services

4.7.3.1 Read

The values of Simple Variable Objects, Arrays and Records of the communication partner may be read by using this service. Single elements of Arrays and Records may be accessed with the subindex.

Table 45. Read

Parameter Name	.req .ind	.res .con
Argument	M	
Access Specification	M	
Index	S	
Variable Name	S	
Variable List Name	S	
Subindex	U	
Result(+)		S
Data		M
Result(-)		S
Error Type		M

Argument
The argument contains the service specific parameters of the service request.

Access Specification
This parameter states if the index or the name is used for addressing.

Index
Logical Address of the object.

Variable Name
This parameter is the name of the object.

Variable List Name
This parameter is the name of the Variable List Object.

Subindex
This parameter contains the logical sub-address of the object.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Data
The data that has been read. The data of the object shall be mapped onto the parameter data according to the description of the Data Types in the coding chapter.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

4.7.3.2 Write

Values are written in the objects Simple Variable, Array, Record and Variable List with this service. Single elements of Arrays and Records may be accessed with the subindex.

Table 46. Write

Parameter Name	.req .ind	.res .con
Argument	M	
Access Specification	M	
Index	S	
Variable Name	S	
Variable List Name	S	
Subindex	U	
Data	M	
Result(+)		S
Result(-)		S
Error Type		M

Argument
The argument contains the service specific parameters of the service request.

Access Specification
This parameter states if the index or the name is used for addressing.

Index
Logical Address of the object.

Variable Name
This parameter is the name of the Variable Object.

Variable List Name
This parameter is the name of the Variable List Object.

Subindex
This parameter contains the logical sub-address of the object.

Data
The data that shall be written. The data of the object shall be mapped onto the parameter data according to the description of the Data Types in the coding chapter.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

4.7.3.3 PhysRead

The value of a Physical Access Object of the communication partner is read using this service.

Table 47. PhysRead

Parameter Name	.req	.res
	.ind	.con
Argument	M	
Local Address	M	
Length	M	
Result(+)		S
Data		M
Result(-)		S
Error Type		M

Argument
The argument contains the service specific parameters of the service request.

Local Address
Physical Address of the object.

Length
Number of octets to be read.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Data
The data that has been read.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

4.7.3.4 PhysWrite

A new value is assigned to a Physical Access Object using this service.

Table 48. PhysWrite

Parameter Name	.req .ind	.res .con
Argument	M	
Local Address	M	
Data	M	
Result(+)		S
Result(-)		S
Error Type		M

Argument
The argument contains the service specific parameters of the service request.

Local Address
Physical Address of the object.

Data
The data to be written.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

4.7.3.5 InformationReport

Values of the objects Simple Variable, Array, Record and Variable List are transmitted by using the InformationReport service. The execution of this service is not confirmed by the communication partner. Single elements of Arrays and Records may be accessed with the subindex.

This service may be mapped in LLI to Layer 2 broadcast or multicast. Thus it is possible to transmit values to all or to some stations in the network.

Table 49. InformationReport

Parameter Name	.req	.ind
Argument	M	
Priority	M	
Access Specification	M	
Index	S	
Variable Name	S	
Variable List Name	S	
Subindex	U	
Data	M	

Argument
The argument contains the service specific parameters of the service request.

Priority
This parameter indicates the priority for this service. It may have the values

true: high priority
false: low priority

Access Specification
This parameter states if the index or the name is used for addressing.

Index
Logical Address of the object in the Source OD.

Variable Name
This parameter is the name of the Variable Object.

Variable List Name
This parameter is the name of the Variable List Object.

Subindex
This parameter is the logical sub-address of the object.

Data
This parameter contains the data. The data of the object shall be mapped onto the parameter data according to the description of the Data Types in the coding chapter.

4.7.3.6 DefineVariableList

A Variable List Object is created at the communication partner using this service. The Number Of Elements attribute is calculated automatically by the server for the DefineVariableList service. The client user shall ensure that the data of the Variable List Service may be transmitted in one single PDU. The Deletable attribute of a Variable List which is created by the DefineVariableList service is set to the value true.

Table 50. DefineVariableList

Parameter Name	.req	.ind	.res	.con
Argument	M			
Password	M			
Access Groups	M			
Access Rights	M			
List Of Variables	M			
Variable Index	S			
Variable Name	S			
Variable List Name	U			
Extension	U			
Result(+)			S	
Index			M	
Result(-)			S	
Error Type			M	

Argument
The argument contains the service specific parameters of the service request. If Access Rights are not supported (Access Protection Supported = false), then the parameters Password, Access Groups and Access Rights are insignificant.

Password
This parameter states to which value the attribute Password shall be set.

Access Groups
This parameter states to which value the attribute Access Groups shall be set.

Access Rights
This parameter states to which value the attribute Access Rights shall be set.

List of Variables
This parameter contains the addresses of the variables which are combined in this Variable List. Each address may be a logical address or a name.

Variable Index
Index of a Variable.

Variable Name
Name of a Variable.

Variable List Name
This parameter states to which value the Name attribute of the Variable List Object shall be set.

Extension
This parameter states to which value the Extension attribute of the Variable List Object shall be set.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Index
Logical Address of the object.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

4.7.3.7 DeleteVariableList

A Variable List Object is deleted at the communication partner with this service if there is an Access Right for this Object. Only Variable List objects having the value true for the Deletable attribute may be deleted.

Table 51. DeleteVariableList

Parameter Name	.req	.res
	.ind	.con
Argument	M	
Access Specification	M	
Index	S	
Variable List Name	S	
Result(+)		S
Result(-)		S
Error Type		M

Argument
The argument contains the service specific parameters of the service request.

Access Specification
This parameter states if the index or the name is used for addressing.

Index
Logical Address of the object.

Variable List Name
This parameter is the name of the Variable List Object.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

4.7.3.8 ReadWithType

The value and the Data Type Description of Simple Variable Objects, Arrays and Records of the communication partner may be read using this service. Single elements of Arrays and Records may be accessed with the subindex.

Table 52. ReadWithType

Parameter Name	.req	.res	.ind	.con
Argument	M			
Access Specification	M			
Index	S			
Variable Name	S			
Variable List Name	S			
Subindex	U			
Result(+)		S		
Type Description		M		
Data		M		
Result(-)		S		
Error Type		M		

Argument
The argument contains the service specific parameters of the service request.

Access Specification
This parameter states if the index or the name is used for addressing.

Index
Logical Address of the object.

Variable Name
This parameter is the name of the Variable Object.

Variable List Name
This parameter is the name of the Variable List Object.

Subindex
This parameter contains the logical sub-address of the object.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Data
The data being read. The data of the object shall be mapped onto the parameter data according to the description of the Data Types in the coding chapter.

Typedescription
This parameter contains the Data Type Description (see Parameter Type Description).

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

4.7.3.9 WriteWithType

Values are written in the objects Simple Variable, Array, Record and Variable List with this service. A Data Type Description is added to the data that is to be written. Single elements of Arrays and Records may be accessed with the subindex.

Table 53. WriteWithType

Parameter Name	.req	.res
	.ind	.con
Argument	M	
Access Specification	M	
Index	S	
Variable Name	S	
Variable List Name	S	
Subindex	U	
Type Description	M	
Data	M	
Result(+)		S
Result(-)		S
Error-Type		M

Argument
The argument contains the service specific parameters of the service request.

Access Specification
This parameter states if the index or the name is used for addressing.

Index
Logical Address of the object.

Variable Name
This parameter is the name of the Variable Object.

Variable List Name
This parameter is the name of the Variable List Object.

Subindex
This parameter contains the logical sub-address of the object.

Typedescription
This parameter contains the Data Type Description.

Data
The data that is to be written. The data of the object shall be mapped onto the parameter data according to the description of the Data Types in the coding chapter.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

4.7.3.10 InformationReportWithType

Values and the Data Type Description of the objects Simple Variable, Array, Record and Variable List are transmitted using the InformationReport service. The execution of this service is not confirmed by the communication partner. Single elements of Arrays and Records may be accessed with the subindex.

This Service may be mapped in LLI to Layer 2 broadcast or multicast. Thus it is possible to transmit values to all or to some stations in the network.

Table 54. InformationReportWithType

Parameter Name	.req	.ind
Argument	M	
Priority	M	
Access Specification	M	
Index	S	
Variable Name	S	
Variable List Name	S	
Subindex	U	
Type Description	M	
Data	M	

Argument
The argument contains the service specific parameters of the service request.

Priority
This parameter indicates the priority for this service. It may have the values

true: high priority
false: low priority

Access Specification
This parameter states if the index or the name is used for addressing.

Index
Logical Address of the object in the Source OD.

Variable Name
This parameter is the name of the Variable Object.

Variable List Name
This parameter is the name of the Variable List Object.

Subindex
This parameter contains the logical sub-address of the object.

Typedescription
This parameter contains the Data Type Description.

Data
This parameter contains the data. The data of the object shall be mapped onto the parameter data according to the description of the Data Types in the coding chapter.

4.8 Event Management

4.8.1 Model Description

The Event serves for sending an important message from one device to another (or in broadcast mode to all other devices). The detection of the conditions that cause an event is responsibility of the user. The application program invokes the EventNotification service when the conditions are met. The monitoring and checking of the (optional) acknowledgement is the responsibility of the user.

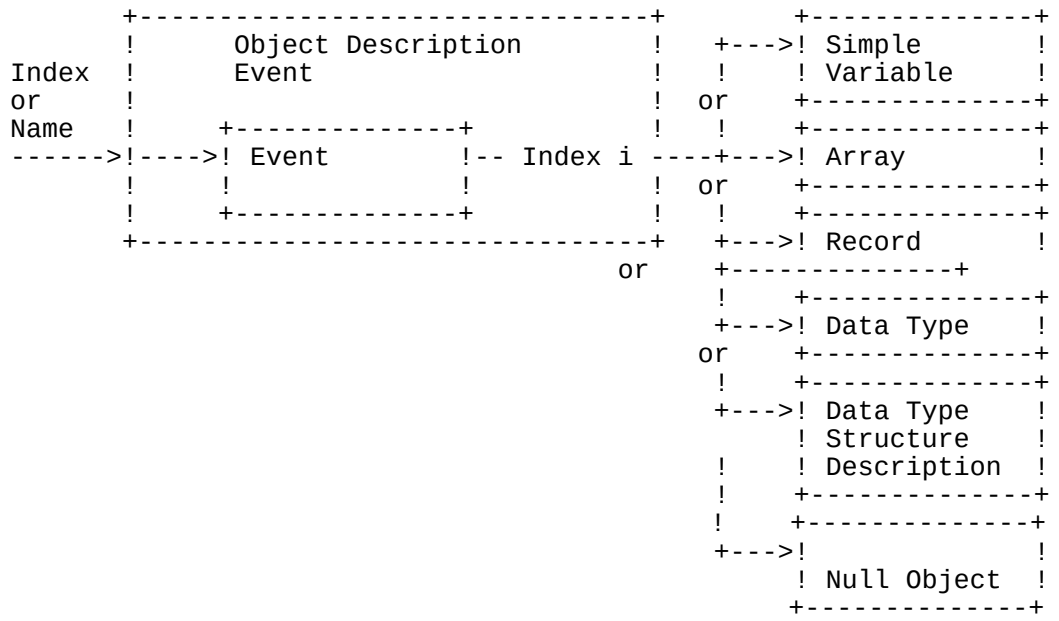


Figure 74. Overview of Event

Services on the Object Event:

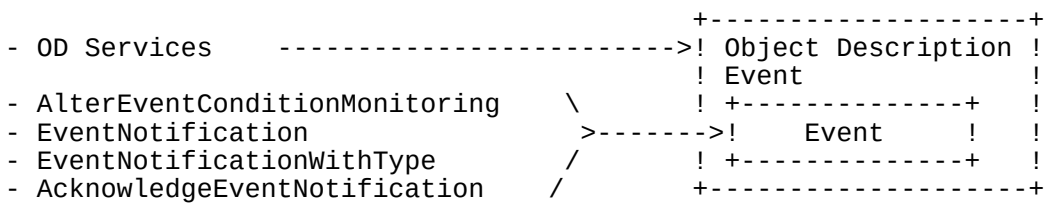


Figure 75. Event Services

An user sends an event notification with the EventNotification service. The user transmits with the EventNotification service a counter value (Event Number) and optionally data (Event Data), e.g. measurement value, status, units.

The management (incrementation) of the counter value is the responsibility of the user. The Event Notification may represent a collective alarm, where the data may contain information about which channel has triggered the collective alarm. Also in this case the logical operations for the triggering conditions and for the data are part of the application program. The receiver of the Event Notification may acknowledge the EventNotification by using the AcknowledgeEventNotification service.

The counter value is transmitted with the AcknowledgeEventNotification service. The counter value serves to reliably correlate the EventNotification and the AcknowledgeEventNotification. The monitoring and the checking of the acknowledgement is the responsibility of the user.

The receiver of the EventNotification may lock or unlock the EventNotification using the AlterEventConditionMonitoring service.

4.8.2 The Event Object

4.8.2.1 Attributes

Object: Event
- Key Attribute: Index
- Key Attribute: Event Name
- Attribute: Index Event Data
- Attribute: Length
- Attribute: Password
- Attribute: Access Groups
- Attribute: Access Rights
- Attribute: Enabled
- Attribute: Extension

Index

Logical Address of the Event Object.

Event Name

The name of the Object. Its existence and its length is defined in the OD Object Description.

Index Event Data

Index for Event Data. This entry defines the index for reading of the Event Data, or it defines under which index the Object Description of the Data Type or the Object Description of the Data Type Structure Description of the Event Data may be read. In the case that no data exist for this event object the Index Event Data shall contain the index of a Null object.

Length

If the Index Event Data points to a Data Type then this attribute contains the length in octets of the Event Data. This attribute is not significant otherwise.

Password

This attribute contains the Password for the Access Rights.

Access Groups

This attribute relates the object to specific Access Groups. The object is a group member when the corresponding bit is set.

Table 55. Access Groups for Event

Bit	Meaning
7	Access Group 1
6	Access Group 2
5	Access Group 3
4	Access Group 4
3	Access Group 5
2	Access Group 6
1	Access Group 7
0	Access Group 8

Access Rights
This attribute contains information about the Access Rights. The specific Access Right exists when the corresponding bit is set.

Table 56. Access Rights for Events

Bit	Name	Meaning
6	W	Right to Acknowledge for the registered Password
5	D	Right to Enable/Disable for the registered Password
2	Wg	Right to Acknowledge for Access Groups
1	Dg	Right to Enable/Disable for Access Groups
14	Wa	Right to Acknowledge for all Communication Partners
13	Da	Right to Enable/Disable for all Communication Partners

Enabled
Status of the Event Object according to the state machine.

Enabled = true <=> UNLOCKED
Enabled = false <=> LOCKED

Extension
This attribute contains profile specific information.

4.8.2.2 Object Description of the OD on Transmission

Index	Object code	Index Event Data	Length	Password	Access Groups	Access Rights	Enabled	Event Name	Extension
130	Event	39	2	17		W D		V. Buding	

Figure 76. Object Description of Event in the S-OD (Example)

Object Code
Tag for the Event Object which is transmitted in addition to the object attributes for the GetOD and the PutOD service.

4.8.3 Event Management Services

4.8.3.1 EventNotification

The EventNotification service allows the transmission of an Event Notification. This service is an unconfirmed service.

Table 57. EventNotification

Parameter Name	.req	.ind
Argument	M	
Priority	M	
Access Specification	M	
Index	S	
Event Name	S	
Event Number	M	
Event Data	O	

Argument
The argument contains the service specific parameters of the service request.

Priority
This parameter defines the priority for this service. It may have the values

true: high priority
false: low priority

Access Specification
This parameter states if the index or the name is used for addressing.

Index
This parameter defines to which Event Object the service is related.

Event Name
This parameter is the name of the Event Object.

Event Number
This parameter contains the counter value.

Event Data
This parameter contains the data. The data of the object shall be mapped onto the parameter data according to the description of the Data Types in the coding chapter. The Index is stored as Index Event Data in the Object Description of the Event Object.

4.8.3.2 AcknowledgeEventNotification

This service allows the acknowledgement of an Event Notification.

Table 58. AcknowledgeEventNotification

Parameter Name	.req	.res
	.ind	.con
Argument	M	
Access Specification	M	
Index	S	
Event Name	S	
Event Number	M	
Result(+)		S
Result(-)		S
Error Type		M

Argument
The argument contains the service specific parameters of the service request.

Access Specification
This parameter states if the index or the name is used for addressing.

Index
This parameter defines to which Event Object the service is related.

Event Name
This parameter is the name of the Event Object.

Event Number
This parameter contains the counter value.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

4.8.3.3 AlterEventConditionMonitoring

This service allows the locking or unlocking of the Event Object.

Table 59. AlterEventConditionMonitoring

Parameter Name	.req .ind	.res .con
Argument	M	
Access Specification	M	
Index	S	
Event Name	S	
Enabled	M	
Result(+)		S
Result(-)		S
Error Type		M

Argument
The argument contains the service specific parameters of the service request.

Access Specification
This parameter states if the index or the name is used for addressing.

Index
This parameter defines to which Event Object the service is related.

Event Name
This parameter is the name of the Event Object.

Enabled
This parameter of type boolean defines to which value the Enabled attribute of the Event Object shall be set.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

Error Type
This parameter contains information about the reason why the service was not completed successfully.

4.8.3.4 EventNotificationWithType

This service allows the transmission of an Event Notification. The notification contains data and the Data Type Description. This service is an unconfirmed service.

Table 60. EventNotificationWithType

Parameter Name	.req	.ind
Argument	M	
Priority	M	
Access Specification	M	
Index	S	
Event Name	S	
Event Number	M	
Type Description	O	
Event Data	O	

Argument
The argument contains the service specific parameters of the service request.

Priority
This parameter defines the priority for this service. It may have the values

true: high priority
false: low priority

Access Specification
This parameter states if the index or the name is used for addressing.

Index
This parameter defines to which Event Object the service is related.

Event Name
This parameter is the name of the Event Object.

Event Number
This parameter contains the counter value.

Typedescription
This parameter contains the Data Type Description (see parameter type description below). It shall be present only if the event data parameter exists.

Event Data
This parameter contains the data. It shall be present only if the Type Description parameter exists. The data of the object shall be mapped onto the parameter data according to the description of the Data Types in the coding chapter. The Index is stored as Index Event Data in the Object Description of the Event Object.

4.8.4 State Machine

4.8.4.1 State Machine Description

LOCKED
No Event Notifications are send in the LOCKED state (Enabled =false).

UNLOCKED
Event Notifications are sent normally in the UNLOCKED state (Enabled =true).

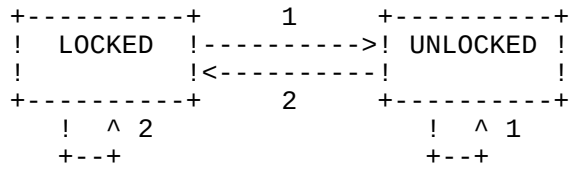
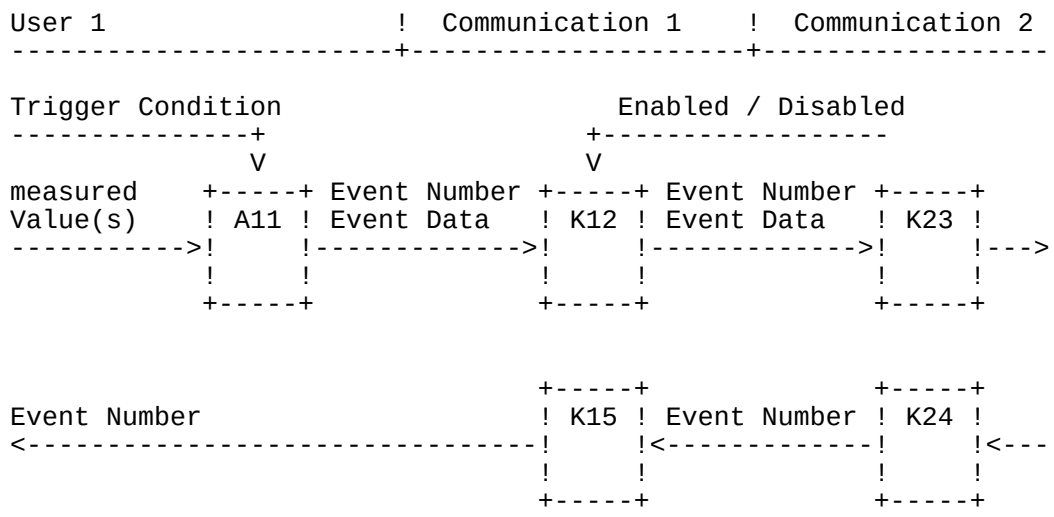


Figure 77. Event State Machine

4.8.4.2 State Transitions

	Event \Exit Condition => Action
1	AlterEventConditionMonitoring.ind \Enabled = true => .res+
2	AlterEventConditionMonitoring.ind \Enabled = false => .res+

4.8.5 EXAMPLE: Event Management Services



A11 Measurement values are compared by User 1 with fixed conditions. If one or more conditions are fulfilled then an EventNotification service is (K12) called. In this service the Event Number and Event Data are transferred as parameters.

K12 -UNLOCKED

The EventNotification service transfers Event Number and Event Data to the communication 2 (K23).

-LOCKED The EventNotification service is disabled. Further actions do not occur.

K23 The communication 2 receives the EventNotification service from communication 1 and passes the Event Number and the Event Data to User 2.

K24 The AcknowledgeEventNotification service is called by User 2. In this service the Event Number is transferred as a parameter. The service transfers the Event Number to the communication 1.

K15 The communication 1 receives the AcknowledgeEventNotification service from communication 2 and passes the Event Number to User 2.

Figure 78. Example Event Services

Trigger conditions:
Any evaluations in the application program that may trigger an Event Notification.

Enabled / Disabled:
The sequencing for the AlterEventConditionMonitoring service is not further described here.

Measurement value(s):
Any signal in the application program that may trigger an event in combination with the trigger condition.

Event Data:
Data transferred as a parameter with the EventNotification service. Data Type and semantics are application specific, e.g.

- Event Code (cause, coming, going)
- Measurement value

- Number of the channel (for collective alarms)
- Time
- Notification lost
- Acknowledgement successful

4.9 Interface between FMS and LLI

The coupling for the FMS to the Lower Layer Interface (LLI) is specified in this section. The LLI provides services to the FMS at its interface which serve to establish and release connections and to transmit data.

4.9.1 Overview of Services

The following services are provided to the FMS:

- Associate (Establishing a connection)
- Abort (Releasing a connection)
- DataTransferConfirmed (Data transmission acknowledged by the user)
- DataTransferAcknowledged (Data transmission acknowledged by LLI)
- DataTransferUnconfirmed (Data transmission without acknowledge)

These services are described by their corresponding service primitives:

- Associate (ASS.req, ASS.ind, ASS.res, ASS.con)
- Abort (ABT.req, ABT.ind)
- DataTransferConfirmed (DTC.req, DTC.ind, DTC.res, DTC.con)
- DataTransferAcknowledged (DTA.req, DTA.ind)
- Data-Transfer-Unconfirmed (DTU.req, DTU.ind)

For more detailed description of the LLI services, their service primitives and their parameters refer to the Lower Layer Interface.

4.9.2 Mapping of FMS Services on LLI Services

The FMS services are mapped onto the services of the Lower Layer Interface in the following way:

- The Initiate service is mapped to the Associate service. The parameter data contains the Initiate PDU.
- The Abort service is mapped directly to the LLI Abort service. No FMS PDU is generated.
- All confirmed FMS services are mapped to the Data Transfer Confirmed Service of the LLI. The parameter data contains the corresponding FMS PDU.
- All Unconfirmed FMS services are mapped to the Data Transfer Acknowledged service of the LLI if the attribute CREL Type of the FMS CRL has the value true. The parameter Data contains the corresponding FMS PDU.
- All Unconfirmed FMS services are mapped to the Data Transfer Unconfirmed service of the LLI if the attribute CREL Type of the FMS CRL has the value false. The parameter Data contains the corresponding FMS PDU.
- The FMS service Reject is mapped to the Service Primitive DTC.res. The parameter Data contains the Reject PDU.

4.10 Interface between FMS and FMA7

The coupling of the FMS to the Fieldbus Management Layer 7 (FMA 7) is described in this section. The FMS provides at this interface services for the FMA7 to configure, to identify and to reset the FMS.

4.10.1 Overview of local FMS Management Services

The following services are provided by FMS to FMA7:

- FMS Disable
- FMS Load CRL
- FMS Enable
- FMS Read CRL
- FMS Ident
- FMS Reset

4.10.1.1 FMS Disable

The data transmissions of the FMS are disabled using this service. All connections are aborted and the FMS user gets an Abort with Reason Code 13. A request for connection establishment from the FMS user or from LLI is rejected with an Abort using Reason Code 13. This service has no parameters and leads always to a positive confirmation.

Table 61. FMS Disable

Parameter Name	.req	.con
Argument	M	
Result(+)		M

Argument
The argument contains no parameter for the service request.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

4.10.1.2 FMS Load CRL

The FMS CRL header or the static part of the FMS CRL Entry of a communication relationship is entered in the FMS CRL using this service.

Table 62. FMS Load CRL

Parameter Name	.req	.con
Argument	M	
FMS CRL Entry Static	M	
Result(+)		S
Result(-)		S
FMA7 Error Type		M

Argument
The argument contains the service specific parameters of the service request.

FMS CRL Entry Static
This parameter contains the FMS CRL Header or the static part of a FMS CRL Entry.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully. In this case no FMS CRL Entry is written into the FMS CRL.

FMA7 Error Type
This parameter contains information about the reason why the service was not completed successfully.

4.10.1.3 FMS Enable

The FMS is enabled using this service. A new connection establishment by the FMS user or the LLI is possible afterwards.

Table 63. FMS Enable

Parameter Name	.req	.con
Argument	M	
Result(+)		S
Result(-)		S
FMA7 Error Type		M
Error CREF		C

Argument
The argument contains no parameter for the service request.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

FMA7 Error Type
This parameter contains information about the reason why the service was not completed successfully.

Error CREF
This parameter defines at which CREF the FMS Enable service is aborted by the FMS. The existence of the parameter depends on the FMA7 Error Type.

4.10.1.4 FMS Read CRL

The FMS CRL Header or a FMS CRL Entry is read using this service.

The Communication Reference 0 shall be used for reading the FMS CRL Header. The corresponding Communication Reference shall be indicated for reading a FMS CRL Entry. If this CREF is not used then this service is answered with a Result(-).

Table 64. FMS Read CRL

Parameter Name	.req	.con
Argument	M	
Desired CREF	M	
Result(+)		S
FMS CRL Entry		M
Result(-)		S
FMA7 Error Type		M

Argument
The argument contains the parameters of the service request.

Desired CREF
For reading the FMS CRL Header the value 0 shall be given, other-wise the CREF of the desired FMS CRL Entry shall be given.

Result(+)
The Result(+) parameter shall indicate that the service was completed success-fully.

FMS CRL Entry
This parameter contains the FMS CRL Header or a FMS CRL Entry.

Result(-)
The Result(-) shall indicate that the service request was not completed success-fully.

FMA7 Error Type
This parameter contains information about the reason why the service was not completed successfully.

4.10.1.5 FMS Ident

This service provides information for identification of the FMS.

Table 65. FMS Ident

Parameter Name	.req	.con
Argument	M	
Result(+)		S
Vendor Name		M
Controller Type		M
Hardware Release		M
Software Release		M
Result(-)		S
FMA7 Error Type		M

Argument
The argument contains no parameter for the service request.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

Vendor Name
This parameter identifies the vendor of the FMS.

Controller Type
This parameter identifies the hardware (eg. Processor, Controller, ASIC) on which the FMS is implemented.

Hardware Release
This parameter identifies the Hardware Release of the FMS.

Software Release
This parameter identifies the Software Release of the FMS.

Result(-)
The Result(-) shall indicate that the service request was not completed successfully.

FMA7 Error Type
This parameter contains information about the reason why the service was not completed successfully.

4.10.1.6 FMS Reset

The FMS is reset using this service. The FMS performs a coldstart after the reset in the same way as after power-up.

Table 66. FMS Reset

Parameter Name	.req	.con
Argument	M	
Result(+)		S

Argument
The argument contains no parameter for the service request.

Result(+)
The Result(+) parameter shall indicate that the service was completed successfully.

4.10.2 FMA7 ErrorType

This parameter contains information about the reason why the service was not completed successfully. The parameter may have the following values:

- No Resource
  This error is indicated when available resources are exceeded.
- No FMS CRL Entry
  No entry was found for the Desired CREF.
- FMS CRL Parameter Invalid
  The FMS CRL Entry contains an illegal value.
- FMS State Conflict
  The service cannot be executed in the current FMS state.
- Other
  The reason for the error is not one of the above

4.11 Operating Behaviour of FMS

The operational behaviour of the FMS is represented by the FMS Basic State Machine.

4.11.1 Start of FMS

After power-up or reset of the FMS by the FMA7 the FMS starts.

The dynamic part of the FMS CRL is created and initialized by the FMS as follows:

If CREL type = true
then OSCC : 0
OSCS : 0
CREL state : CONNECTION-NOT-ESTABLISHED

If CREL type = false AND Max-PDU-Receiving-High-Prio = 0 AND Max-PDU-Receiving-Low-Prio = 0

then OSCC : 0
OSCS : 0
CREL state : CONNECTIONLESS-CLIENT

If CREL type = false AND Max-PDU-RSending-High-Prio = 0 AND Max-PDU-Sending-Low-Prio = 0

then OSCC : 0
OSCS : 0
CREL state : CONNECTIONLESS-SERVER

4.11.2 Conditions of Readiness for Operation

For the successful termination of the starting phase (State of the FMS Basic State Machine = FMS READY) the following conditions shall be fulfilled:

- Static part of the FMS CRL exists
- Sufficient resources for the dynamic part of the FMS CRL are available

4.11.3 FMS Readiness for Operation

In the state FMS READY the FMS is operational and may accept connection establishment requests or data transfer requests (broadcast / multicast).

4.11.4 FMS Basic State Machine

4.11.4.1 FMS Basic State Machine Description

FMS START
The current FMS CRL is checked. This state is only exited when a valid FMS CRL exists.

FMS DISABLE
FMS is disabled by the FMA7 for data transmission.

LOADING CRL
In this state the FMS CRL is loaded with FMS Load CRL services.

CHECK CRL
The FMS CRL is checked after a loading sequence.

FMS READY
FMS is ready for data transmission.

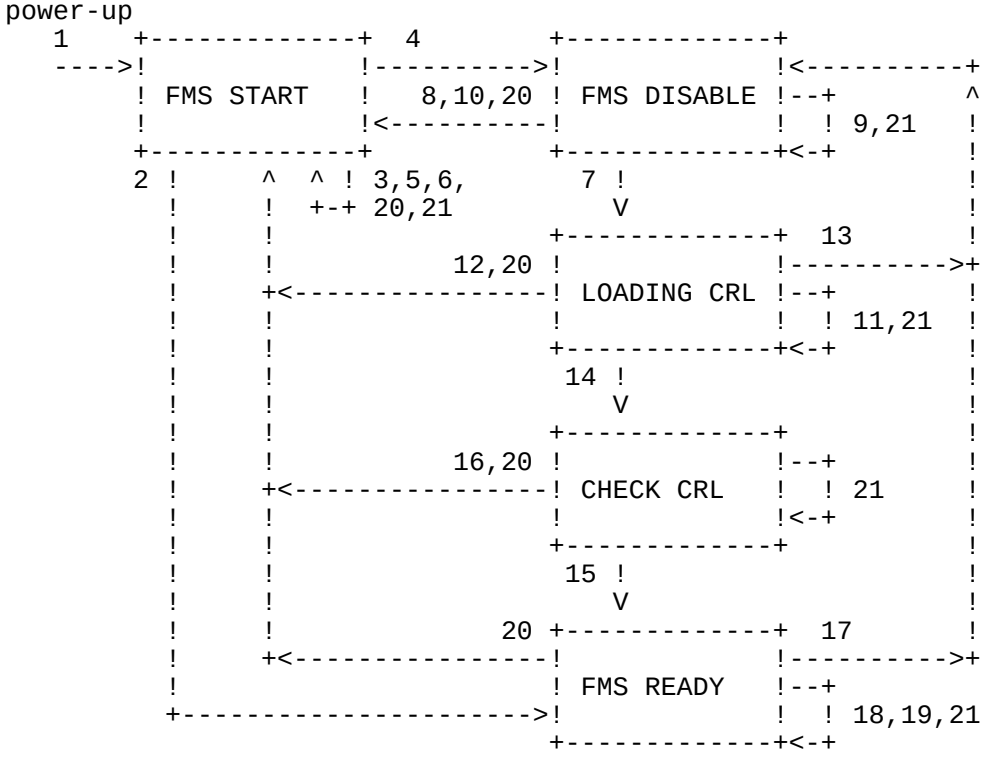


Figure 79. FMS Basic State Machine

4.11.4.2 State Transitions

Current State Event \Exit Condition => Action Taken	Transition	Next State
Power-On => Start FMS CRL test	**1**	**FMS-START**
FMS-START FMS CRL test finished \valid FMS CRL available AND resources sufficient => dynamic part of the FMS CRL preset start all of FMS state machines related to CREF	**2**	**FMS-READY**
FMS-START FMS CRL test finished \no valid FMS CRL available OR resources not sufficient	**3**	**FMS-START**
FMS-START FMS-Disable.req => FMS-Disable.con(+)	**4**	**FMS-DISABLE**

Current State Event \Exit Condition => Action Taken	Transition	Next State
FMS-START FMS-Enable.req => FMS-Enable.con(-)	**5**	**FMS-START**
FMS-START FMS-Load-CRL.req => FMS-Load-CRL.con(-)	**6**	**FMS-START**
FMS-START any service primitive from the FMS user => Abort.ind (ABT RC13)	**23**	**FMS-START**
FMS-START any service primitive from the LLI => ABT.req (ABT RC13)	**24**	**FMS-START**
FMS-DISABLE FMS-Load-CRL.req \FMS CRL entry valid => FMS-Load-CRL.con(+)	**7**	**LOADING-CRL**
FMS-DISABLE FMS-Load-CRL.req \FMS CRL entry not valid => FMS-Load-CRL.con(-)	**8**	**FMS-START**
FMS-DISABLE FMS-Disable.req => FMS-Disable.con(+)	**9**	**FMS-DISABLE**
FMS-DISABLE FMS-Enable.req => FMS-Enable.con(-)	**10**	**FMS-START**
FMS-DISABLE any service primitive from the FMS user => Abort.ind (ABT RC13)	**25**	**FMS-DISABLE**
FMS-DISABLE any service primitive from the LLI => ABT.req (ABT RC13)	**26**	**FMS-DISABLE**
LOADING-CRL FMS-Load-CRL.req \FMS CRL entry valid => FMS-Load-CRL.con(+)	**11**	**LOADING-CRL**
LOADING-CRL FMS-Load CRL.req \FMS CRL entry not valid => FMS-Load-CRL.con(-)	**12**	**FMS-START**

Current State Event \Exit Condition => Action Taken	Transition	Next State
LOADING-CRL FMS-Disable.req => FMS-Disable.con(+)	**13**	**FMS-DISABLE**
LOADING-CRL FMS-Enable.req => Start FMS CRL test	**14**	**CHECK-CRL**
LOADING-CRL any service primitive from the FMS user => Abort.ind (ABT RC13)	**27**	**LOADING-CRL**
LOADING-CRL any service primitive from the LLI => ABT.req (ABT RC13)	**28**	**LOADING-CRL**
CHECK-CRL FMS CRL test finished \valid FMS CRL available AND resources sufficient => dynamic part of the FMS CRL preset start all of FMS state machines related to CREF FMS-Enable.con(+)	**15**	**FMS-READY**
CHECK-CRL FMS CRL test finished \no valid FMS CRL available OR resources not sufficient => FMS-Enable.con(-)	**16**	**FMS-START**
CHECK-CRL any service primitive from the FMS user => Abort.ind (ABT RC13)	**29**	**CHECK-CRL**
CHECK-CRL any service primitive from the LLI => ABT.req (ABT RC13)	**30**	**CHECK-CRL**
FMS-READY FMS-Disable.req => release all FMS connections ABT.req(ABT_RC13) ABT.ind(ABT RC13) FMS-Disable.con(+)	**17**	**FMS-DISABLE**
FMS-READY FMS-Enable.req => FMS-Enable.con(-)	**18**	**FMS-READY**
FMS-READY FMS-Load-CRL.req => FMS-Load-CRL.con(-)	**19**	**FMS-READY**

Current State Event \Exit Condition => Action Taken	Transition	Next State
FMS-READY any Service primitive with no related CREF received from the FMS user => Abort.ind <ABT_RC2>	**22**	**FMS-READY**
FMS-READY any Service primitive with no related CREF received from the LLI => ABT.req <ABT_RC5>	**31**	**FMS-READY**
<any State> FMS-Reset.req => reset all CREFs start FMS CRL test FMS-Reset.con(+)	**20**	**FMS-START**
<any State> FMS-Ident.req OR FMS-Read-CRL.req => execute FMS service FMS Service.con(+/-) an FMA7	**21**	**<same State>**

4.12 Structured Parameters ErrorType and TypeDescription

4.12.1 Parameter Error Type

An error message contained in Result(-) has the following structure.

Table 67. Error Type

Error Type	.res	.con
Error Class		M	M
Error Code		M	M
Additional Code		U	U
Additional Description		U	U

Error Class
This parameter indicates the kind of error. For coding see section 3.16 Syntax Description.

Error Code
This parameter gives a more detailed specification of the error. For Coding see Syntax Description.

Additional Code (optional)
This parameter is optional. The user is responsible for its usage and evaluation.

Additional Description (optional)
This parameter is optional and may be used to add a text to the error message.

4.12.1.1 Meaning of Error Class and Error Code

Error Class

Meaning of the Error Class

- Error Code

  Meaning of the Error Code

VFD State

This error class is returned whenever the state of the VFD is such that the request service may not be executed.

- Other

  This Error Code is returned due to a reason other than any of those listed above.

Application Reference

This Error Class is related to the communication relationship on which the service is executed.

- Application-Unreachable
  The application process is not reachable.
- Other
  This Error Code is returned due to a reason other than any of those listed above.

Definition

This Error Class is returned when there are problems with object definitions (CreateProgramInvocation, DefineVariableList).

- Object-Undefined
  The required objects do not exist.
- Object-Attribute-Inconsistent
  The indicated objects are defined with inconsistent attributes
- Name-Already-exists
  The name exists already.
- Other
  This Error Code is returned due to a reason other than any of those listed above.

Resource

This Error Class is returned when available resources are exceeded.

- Memory-Unavailable
  There is no more memory for the execution of this service.
- Other
  This Error Code is returned due to a reason other than any of those listed above.

Error Class

Meaning of the Error Class
- Error Code
  Meaning of the Error Code

Service

This Error Class is returned whenever there are problems with the service itself.

- Object-State-Conflict
  The current state of the object does not permit execution of the service.
- Object-Constraint-Conflict
  The execution of the service is not possible at this time.
- Parameter-Inconsistent
  The service contains inconsistent parameters.
- Illegal-Parameter
  A parameter has an illegal value.
- PDU-Size
  Response-PDU > Max-PDU-Sending-Low-Prio.

- Other
  This Error Code is returned due to a reason other than any of those listed above.

Access

This Error Class is returned whenever an access is faulty.

- Object-Access-Unsupported

  The object has not been defined for the requested access.

- Object-Non-Existent
  The object does not exist.

- Object-Access-Denied
  The FMS client has not sufficient Access Rights for the object.

- Invalid-Address
  The indicated Address is out of the legal range (PhysRead, PhysWrite)

- Object-Invalidated
  The access refers to a defined object with an undefined Reference attribute. This is a permanent error.

- Hardware-Fault
  The access to the object failed because of a hard-ware error.

Error Class

Meaning of the Error Class
- Error Code
  Meaning of the Error Code

Access
- Type-Conflict
  The access is rejected because of an incorrect Data Type.

- Named-Access-Unsupported
  The access with names is not supported.

- Object-Attribute-inconsistent
  The attributes of the objects are inconsistent.

- Other
  This Error Code is returned due to a reason other than any of those listed above.

OD-Error

This Error Class is returned whenever an incorrect change to the OD is made (PutOD, CreateProgramInvocation, DefineVariableList).

- Name-Length-Overflow
  The legal length of the name is exceeded.

- OD-Overflow
  The legal length of the OD is exceeded.

- OD-Write-Protected
  The Object Dictionary is write protected.
- Extension-Length-Overflow
  The legal length for the Extension is exceeded.
- Object-Description-Length-Overflow
  The legal length of a single Object Description is exceeded.
- Operational-Problem
  The currently loaded OD is incorrect.
- Other
  This Error Code is returned due to a reason other than any of those listed above.

Others

This Error Class is returned due to a reason other than any of those listed above.

- Other
  This Error Code is returned due to a reason other than any of those listed above.

4.12.1.2 Meaning of the remaining Parameters

Additional Code
no further specification

Additional Description
no further specification

4.12.2 Parameter Type Description

Table 68. Type Description

Parameter Name	M
Type Description	M
Simple	S
Data Type Index	M
Length	M
Array	S
Data Type Index	M
Length	M
Number Of Elements	M
Structure	S
List Of Elements Type	M
Data Type Index	M
Length	M

Typedescription
This parameter contains the Data Type Description.

Simple
This parameter contains the Data Type and Length of a Simple Variable.

Array
This parameter contains Data Type and Length of an element of the array and the number of elements.

Structure
This parameter contains a list of Data Type Descriptions of the Record Elements.

List-Of-Elements-Type
This parameter is the list of Data Type Descriptions of the Record Elements. The Data Type and the Length is given for each element.

Data-Type-Index
Logical Address of the Data Type

Length
Length in octets

4.13 Tables

4.13.1 List of Object Codes

Table 69. Object Codes

Object	Object Code
Null Object	0
- reserved -	1
Domain	2
Program Invocation	3
Event	4
Data Type	5
Data Type Structure Description	6
Simple Variable	7
Array	8
Record	9
Variable List	10

The Object Code shall be of Type Integer8.

4.13.2 List of Standard Data Types

Table 70. Standard Data Types

Data Type	Index	Number of Octets
Boolean	1	1
Integer8	2	1
Integer16	3	2
Integer32	4	4
Unsigned8	5	1
Unsigned16	6	2
Unsigned32	7	4
Floating Point	8	4
Visible String	9	1, 2, 3..
Octet String	10	1, 2, 3..
Date	11	7
Time Of Day	12	4 or 6
Time Difference	13	4 or 6
Bit String	14	1, 2, 3..

4.13.3 List of Object Attributes and Service Parameters

Table 71. Object Attributes and Service Parameters

Object Attribute / Service Parameter	Data Type
Abort Detail	Octet String, max. 16 octets
Abort Identifier	Unsigned8
Access Groups	Bit String, 1 Octet
Access Protection Supported	Boolean
Access Rights	Bit String, 2 Octets
Access Specification	--
Additional Code	Integer16
Additional Description	Visible String
Additional Information	Visible String
All Attributes	Boolean
Number FMS CRL Entries	Integer16
Array	--
Consequence	Integer8
Controller Type	Visible String
Counter	Integer8
Data	--
Data Type Index	Unsigned16
Deletable	Boolean
Description	--
Detected Here	Boolean
Domain Index	Unsigned16
Domain Name	Visible String,max 32 Octets
Domain State	Unsigned8
DP-OD Length	Unsigned16
DV-OD Length	Unsigned16
Element Index	Unsigned16
Enabled	Boolean
Error Class	Integer8
Error Code	Integer8
Error CREF	Unsigned16
Error Type	--
Event Data	Octet String
Event Name	Visible String,max.32 octets
Event Number	Unsigned8
Execution Argument	Octet String
Extension	Length + Octet String
Final Result	Boolean
First Index DP-OD	Unsigned16
First Index DV-OD	Unsigned16
First Index S-OD	Unsigned16
FMS CRL Entry	Octet String
FMS CRL Entry Static	Octet String
FMS Features Supported	Bit String, 6 Octets
Hardware Release	Visible String
Index	Unsigned16
Index Event Data	Unsigned16
Invoke ID	Integer8
CREL State	Unsigned8
CREL Type	Boolean
CREF	Unsigned16
Length	Unsigned8
List Of Elements Type	--
List of Element Index	--
List of Domains	--
List of Index	--
List of Local Address	--
List of Object Description	--

Table 70. (Continuation)

Object Attribute / Service Parameter	Data Type
List of Variables	!--
List of VFD Specific Objects	!--
Load Data	!Octet String
Local Address	!Unsigned32
Local Address DP-OD	!Unsigned32
Local Address DV-OD	!Unsigned32
Local Address OD-ODES	!Unsigned32
Local Address S-OD	!Unsigned32
Local Address ST-OD	!Unsigned32
Local Detail	!Bit String, 3 Octets
Locally Generated	!Boolean
Logical Status	!Unsigned8
Max Octets	!Unsigned16
Max Outstanding Services Client	!Unsigned8
Max Outstanding Services Server	!Unsigned8
Max PDU Receiving High Prio	!Unsigned8
Max PDU Receiving Low Prio	!Unsigned8
Max PDU Sending High Prio	!Unsigned8
Max PDU Sending Low Prio	!Unsigned8
Model Name	!Visible String
More Follows	!Boolean
Name Length	!Unsigned8
Number Of Domains	!Unsigned8
Number Of Elements	!Unsigned8
Original Invoke ID	!Integer8
Outstanding Services Counter Client	!Unsigned8
Outstanding Services Counter Server	!Unsigned8
OD Object Description	!Octet String
Password	!Unsigned8
Physical Status	!Unsigned8
PI Name	!Visible String,max.32 octets
PI State	!Unsigned8
Priority	!Boolean
Profile Number	!Octet String, 2 octets
Reason Code	!Unsigned8
Reject Code	!Integer8
Reject PDU Type	!Unsigned8
Reusable	!Boolean
Revision	!Visible String
ROM/RAM Flag	!Boolean
S-OD Length	!Unsigned16
Simple	!--
Software Release	!Visible String
ST-OD Length	!Unsigned16
Startindex	!Unsigned16
Structure	!--
Subindex	!Unsigned8
Symbol	!Visible String
Symbol Length	!Unsigned8
Type Description	!--
Upload State	!Unsigned8
Variable Index	!Unsigned16
Variable List Name	!Visible String,max.32 octets
Variable Name	!Visible String,max.32 octets
Vendor Name	!Visible String
Version OD	!Integer16
VFD Pointer	!Unsigned32
VFD Pointer Supported	!Boolean

4.13.4 List of Services

Table 72. Meaning of Services

Service	Meaning
Initiate	Establish a connection
Abort	Release a connection
Reject	Reject an improper service or PDU
Status	Read a device / user status
UnsolicitedStatus	Report an unsolicited status
Identify	Read vendor, type and version
GetOD	Read an Object Description
InitiatePutOD	Start an OD Load
PutOD	Load an Object Description
TerminatePutOD	Stop an OD Load
InitiateDownloadSequence	Open a Download sequence
DownloadSegment	Transmit a Download Data block
TerminateDownloadSequence	Stop a Download sequence
RequestDomainDownload	Request a Download
InitiateUploadSequence	Open an Upload sequence
UploadSegment	Transmit an Upload data block
TerminateUploadSequence	Stop an Upload sequence
RequestDomainUpload	Request an Upload
CreateProgramInvocation	Create a program
DeleteProgramInvocation	Delete a program
Start	Start a program - after Reset
Stop	Stop a program
Resume	Resume a program - after Stop
Reset	Reset program
Kill	Kill a program
Read	Read a variable
Write	Write a variable
ReadWithType	Read a variable with Type
WriteWithType	Write a variable with Type
PhysRead	Read a memory location
PhysWrite	Write a memory location
InformationReport	Send Data
InformationReportWithType	Send Data with Type
DefineVariableList	Define a Variable List
DeleteVariableList	Delete a Variable List
EventNotification	Report an event
EventNotificationWithType	Report an event with Type
AcknowledgeEventNotification	Acknowledge an event
AlterEventConditionMonitoring	Disable / Enable an event

4.14 Conformance

This section describes the Protocol Implementation Conformance Statements (PICS). Every vendor of a device shall fill out these tables completely.

There are 4 parts of the PICS:

- Part 1 states information about the implementation and system
- Part 2 is a list of the supported services
- Part 3 is a list of the supported parameters
- Part 4 is a list of the local implementation values

4.14.1 Implementation and System (PICS Part 1)

Table 73. Implementation and System

System Parameters	Detail
Implementation's Vendor Name	
Implementation's Model Name	
Implementation's Revision Identifier	
Vendor Name of FMS	
Controller Type of FMS	
Hardware Release of FMS	
Software Release of FMS	
Profile Number	
Calling FMS User (enter "Yes" or "No")	
Called FMS User (enter "Yes" or "No")	

4.14.2 Supported Services (PICS Part 2)

Support of the service primitive ("Yes" or "No") shall be entered in every line. If there is more than one service primitive in one line then "Yes" may be entered only if all service primitives listed in this line are supported.

Table 74. Supported Services

Service	Primitive Supported ?	
Initiate	.req,.con	
Status	.req,.con	
Identify	.req,.con	
GetOD	.req,.con	
GetOD (Long Form)	.req,.con	.ind,.res
UnsolicitedStatus	.req	.ind
InitiatePutOD	.req,.con	.ind,.res
PutOD	.req,.con	.ind,.res
TerminatePutOD	.req,.con	.ind,.res

Table 73. (Continuation)

Service	Primitive Supported?	
InitiateDownloadSequence	.req,.con	.ind,.res
DownloadSegment	.ind,.res	.req,.con
TerminateDownloadSequence	.req,.con	.ind,.res
InitiateUploadSequence	.req,.con	.ind,.res
UploadSegment	.req,.con	.ind,.res
TerminateUploadSequence	.req,.con	.ind,.res
RequestDomainDownload	.req,.con	.ind,.res
RequestDomainUpload	.req,.con	.ind,.res
CreateProgramInvocation	.req,.con	.ind,.res
DeleteProgramInvocation	.req,.con	.ind,.res
Start	.req,.con	.ind,.res
Stop	.req,.con	.ind,.res
Resume	.req,.con	.ind,.res
Reset	.req,.con	.ind,.res
Kill	.req,.con	.ind,.res
Read	.req,.con	.ind,.res
Write	.req,.con	.ind,.res
ReadWithType	.req,.con	.ind,.res
WriteWithType	.req,.con	.ind,.res
PhysRead	.req,.con	.ind,.res
PhysWrite	.req,.con	.ind,.res
InformationReport	.req	.ind
InformationReportWithType	.req	.ind
DefineVariableList	.req,.con	.ind,.res
DeleteVariableList	.req,.con	.ind,.res
EventNotification	.req	.ind
EventNotificationWithType	.req	.ind
AcknowledgeEventNotification	.req,.con	.ind,.res
AlterEventConditionMonitoring	.req,.con	.ind,.res

4.14.3 FMS Parameters and Options (PICS Part 3)

The vendor of a device shall indicate with a "Yes" or a "No" in each line if the FMS implementation supports the parameter or the option.

Table 75. FMS Parameters and Options

FMS Parameters and Options	Detail
Addressing by names	yes/no
Maximum length for names	Value
Access Protection supported	yes/no
Maximum length for Extension	Value
Maximum length for Execution Arguments	Value

4.14.4 Local Implementation Values (PICS Part 4)

Table 76. Local Implementation Values

Local Implementation Values	Detail
Maximum length of a FMS PDU	Value
Maximum number of Services Outstanding Calling	Value
Maximum number of Services Outstanding Called	Value
Syntax and semantics of the Execution Argument	Explanation
Syntax and semantics of extension	Explanation

PROFIBUS Specification - Normative Parts

Part 6

Application Layer Protocol Specification

CONTENTS

1 Scope

(see 5)

2 Normative References and additional Material

(see 5)

3 General

(see 5)

4 Coding

4.1 General

Additional information is to be added to the user data to allow an unique association of the data at the communication partner. The coding rules for the additional information are optimized to produce messages as short as possible in accordance with fieldbus requirements. The frequency of occurrence of special messages is taken into account.

The conditions in the fieldbus area are the following:

- short messages
- low number of different messages
- some messages such as Read and Write occur especially often

4.2 Coding Rules

The structuring of a FMS PDU is done either by inserting explicitly Identification Information or by implicit agreements.

Structure of a PDU:

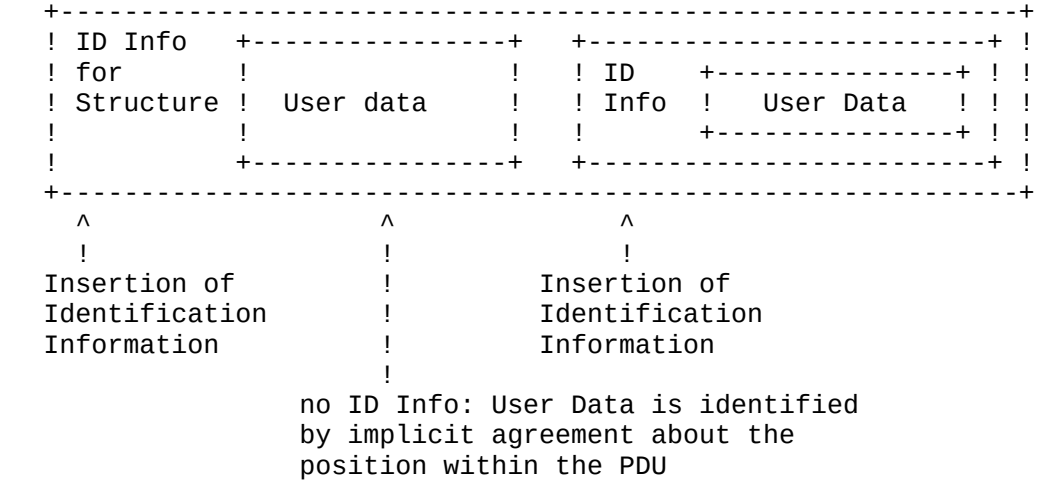


Figure 1. Insertion of Identification Information in the FMS PDU

The Identification Information consists of P/C flag, tag and length. The structures and the user data of the PDU may be identified using this Identification Information.

The semantics of the user data are either known from the context or are universally known (context specific tags or universal tags). In the syntax description the context specific tags are enclosed in rectangular brackets. If the semantics of a parameter are implicitly known from the position in the PDU, no tag is used.

The following restrictions on the usage of implicitly known components (universal tags) shall be made:

- The length of the user data shall be fixed
- The component may not be OPTIONAL
- The component may not be inside a CHOICE construct

4.3 Structure of the Identification Information

The Identification Information (ID Info) consists of P/C flag, tag and length.

The P/C flag indicates if it is a simple component (primitive) or if it is a structured component (constructed, see also subclause 3.15.3.2 - SEQUENCE, SEQUENCE OF).

```
P/C Flag =0  <=>  simple component
P/C Flag =1  <=>  structured component
```

The tag identifies the semantics of the component.

The length is the length of the component in octets if this component is a simple one or the number of contained components if this component is structured.

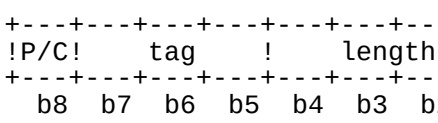
The whole ID Info is coded in one octet, if possible. The octet is divided into 3 parts of different lengths.

- P/C flag 1 bit
- tag 3 bit
- length 4 bit.

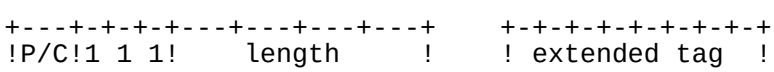
If the space in the fields for the length or for the tag is not sufficient, an extension is used. For this all bits of the concerned field are set to one. The information is then encoded in the following octet. The range of the tag is 0 to 6 and the range of the length is 0 to 14 when no extension is used. These ranges are preferred for the most frequently occurring messages because they produce very short PDUs.

If an extension has to be used for both the tag and the length, the tag is encoded in the first subsequent octet and the length is encoded in the second subsequent octet.

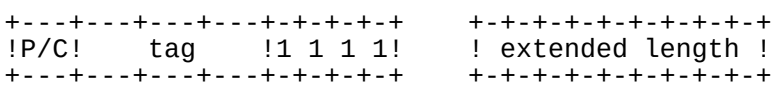
a) Tag 0 to 6, Length 0 to 14



b) Tag 7 to 255, Length 0 to 14



c) Tag 0 to 6, Length 15 to 255



d) Tag 7 to 255, Length 15 to 255

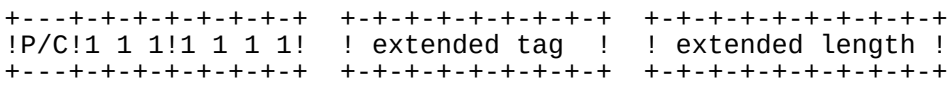
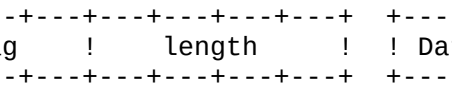


Figure 2. Coding of the ID Info with and without Extension

4.3.1 Coding of Application Data

User data is always a simple (primitive) component. It is encoded in the following way:



If the semantics of the user data are known implicitly from the position in the PDU and the length is fixed and implicitly known, then no Identification Information is added.

4.3.1.1 Boolean

Representation of the value true or false in one octet:

Notation: Boolean
Range of values: true or false
Coding: false is represented by the value 0,
true is represented by the value FF

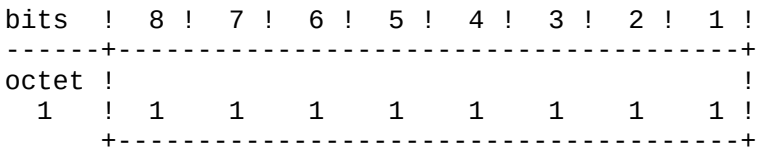


Figure 3. Representation of the Value true

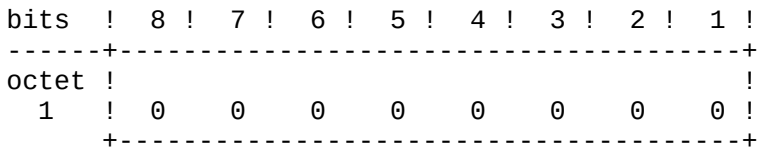


Figure 4. Representation of the Value false

4.3.1.2 Integer

Integer values are signed quantities.

Notation: Integer8, Integer16, Integer32

Range of Values:

Data Type	range of values	length
Integer8	-128 • i • 127	1 octet
Integer16	-32768 • i • 32767	2 octets
	31 31	
Integer32	-2 • i • 2 - 1	4 octets

Coding: In two's complement representation; the MSB (Most Significant Bit)is the bit after the sign bit (SN) in the first octet.
SN = 0: positive numbers and zero
SN = 1: negative numbers

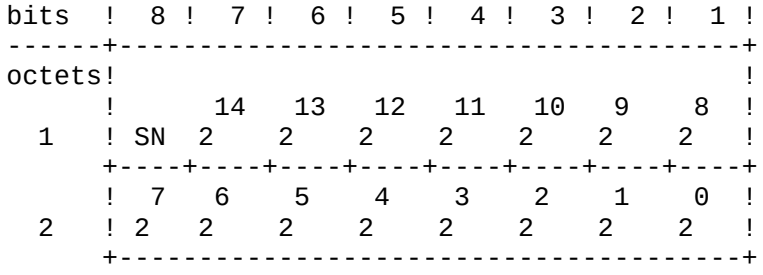


Figure 5. Coding of Data of Data Type Integer16

4.3.1.3 Unsigned

Unsigned Values.

Notation: Unsigned8, Unsigned16, Unsigned32

Range of Values:

Data Type	range of values	length
Unsigned8	0 • i • 255	1 octet
Unsigned16	0 • i • 65535	2 octets
Unsigned32	0 • i • 4294967295	4 octets

Coding: Binary

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

Figure 6. Coding of Data of Data Type Unsigned16

4.3.1.4 Floating Point

Floating Point Number

Notation: Floating-Point (4 octets)
Range of Values: see IEEE Std 754 Short Real Number (32 bits)
Coding: see IEEE Std 754 Short Real Number (32 bits)

								LSB
bits	8	7	6	5	4	3	2	1
octets	Exponent (E)							
		7	6	5	4	3	2	1
1	SN	2	2	2	2	2	2	2
	(E)	Fraction (F)						
	0	-1	-2	-3	-4	-5	-6	-7
2	2	2	2	2	2	2	2	2
	Fraction (F)							
	-8	-9	-10	-11	-12	-13	-14	-15
3	2	2	2	2	2	2	2	2
	Fraction (F)							
	-16	-17	-18	-19	-20	-21	-22	-23
4	2	2	2	2	2	2	2	2

SN: sign 0 = positive, 1 = negative

Figure 7. Coding of Data of Data Type Floating Point

4.3.1.5 Visible String

Notation: Visible-String
Range of Values: see ISO 646 and ISO 2375: Defining registration number 2 + SPACE

Coding: see ISO 646

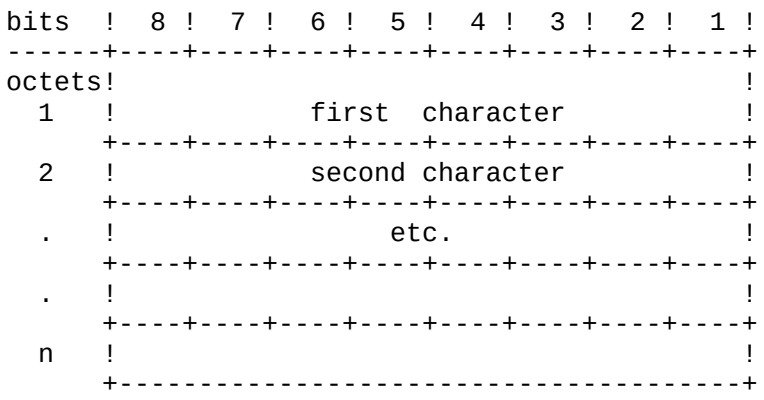


Figure 8. Coding of Data of the Data Type Visible String

4.3.1.6 Octet String

Notation: Octet String

Coding: Binary

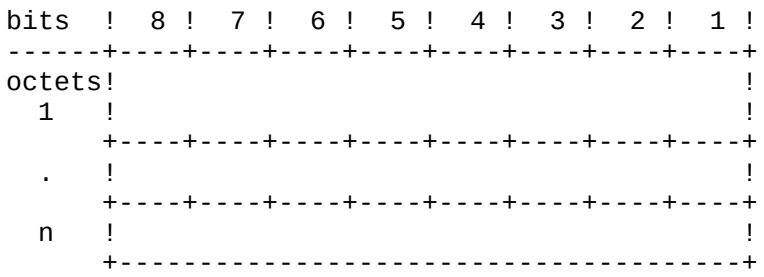


Figure 9. Coding of Data of the Data Type Octet String

4.3.1.7 Date

The Data Type Date consists of a calendar date and a time.
Notation: Date / Time
Range of Values: ms to 99 years
Coding: in 7 octets

Parameter	Range of Values	Meaning of the Parameters
ms	0...59 999	milli-seconds
min	0...59	minutes
SU	0,1	0: standard time, 1: summer time
RSV		reserve
h	0...23	hours
d. of w.	1...7	day of week: 1 = Monday, 7 = Sunday
d. of m.	1...31	day of month
months	1...12	months
years	0...99	years (without the century)

bits	8	7	6	5	4	3	2	1	
octets	15	14	13	12	11	10	9	8	
1	2	2	2	2	2	2	2	2	
	7	6	5	4	3	2	1	0	0...59 999 ms
2	2	2	2	2	2	2	2	2	
			5	4	3	2	1	0	0...59 min
3	RSV	RSV	2	2	2	2	2	2	
			4	3	2	1	0		0...23 hours
4	SU	RSV	RSV	2	2	2	2	2	
	day of week			day of month					1...7 d. of w.
	2	1	0	4	3	2	1	0	
5	2	2	2	2	2	2	2	2	1...31 d. of m.
			5	4	3	2	1	0	1...12 months
6	RSV	RSV	2	2	2	2	2	2	
		6	5	4	3	2	1	0	0 ... 99 years
7	RSV	2	2	2	2	2	2	2	

MSB

Figure 10. Coding of Data of the Data Type Date

4.3.1.8 Time Of Day

The Data Type Time Of Day consists of a time and an optional date.

The time is stated in milliseconds since midnight. At midnight the counting starts with the value zero.

The date is stated in days relatively to the first of January 1984. On the first of January 1984 the date starts with the value zero.

Notation: Time Of Day
Range of Values: $0 \bullet i \bullet (2^{28} - 1)$ ms
$0 \bullet i \bullet (2^{16} - 1)$ days

Coding:

The time is represented as a 32 bit binary value. The first four (MSB) bits shall have the value zero.

The (optional) date is encoded as a 16 bit (2 octets) binary value.

The Time Of Day is a string of 4 or 6 octets.

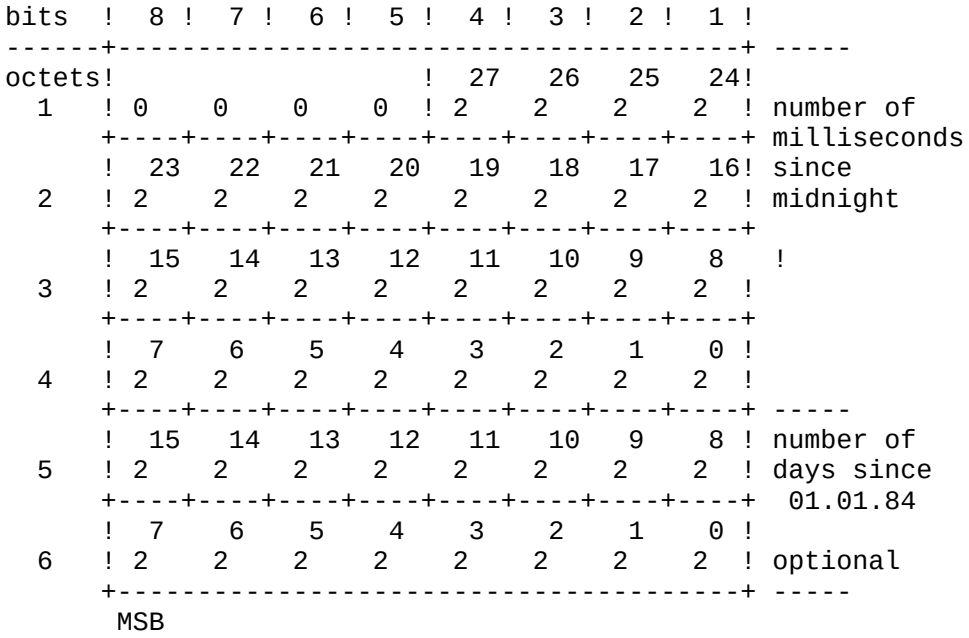


Figure 11. Coding of Data of the Data Type Time Of Day

4.3.1.9 Time Difference

The Data Type Time Difference consists of a time in milliseconds and an optional day count. The structure is equivalent to the structure of the Time Of Day but it states in this case a Time Difference.

Notation: Time Difference
Range of Values: $0 \bullet i \bullet (2^{32} - 1)$ ms
$0 \bullet i \bullet (2^{16} - 1)$ days

Coding:

The time is represented as a 32 bit binary value. The first four (MSB) bits shall have the value zero.

The (optional) date is encoded as a 16 bit (2 octets) binary value.

The Time Difference is a string of 4 or 6 octets.

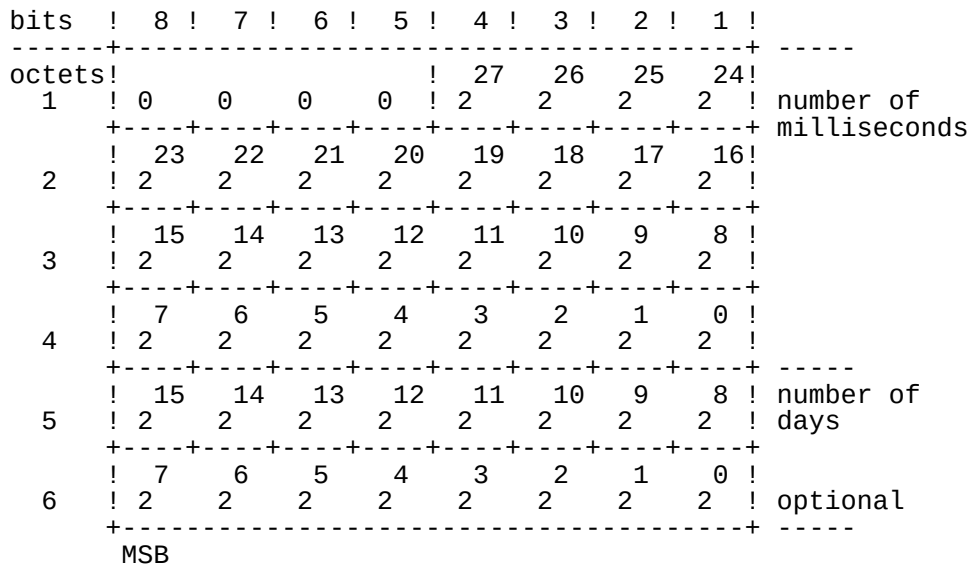


Figure 12. Coding of Data of The Data Type Time Difference

4.3.1.10 Bit String

The following figure shows the numbering scheme of the bits of the Data Type Bit String.

The length of the Bit String shall be given in octets. Therefore only multibles of eight bits can be coded as a Bit String.

Notation: Bit String

Coding: Binary

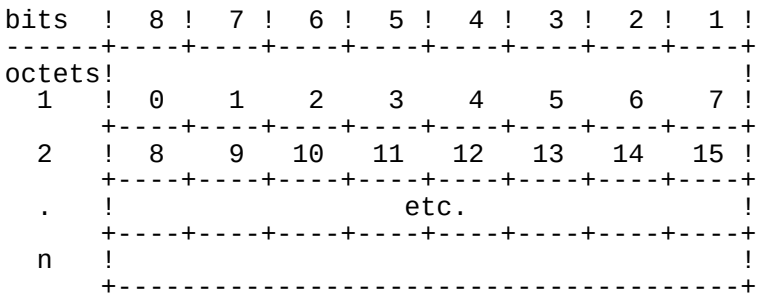


Figure 13. Coding of Data of the Data Type Bit String

4.3.1.11 Null

The Data Type Null has the length zero. There are no subsequent (empty) octets.

4.3.1.12 Packed

The Data Type Packed contains one or more data elements of the Data Types described in the coding of application data chapter which are chained together without any gap. The composition is known implicitly.

4.3.2 Coding of Structure Information

User data may be combined to structured (constructed) components.

The communication partner shall be able to identify these structures and the components of these structures. The P/C flag of the ID Info is 1.

4.3.2.1 SEQUENCE

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

Before the SEQUENCE structure there is an ID Info, which conveys the length not in octets but in number of components. The number of components is less than the total length in octets. In most cases an extension is not necessary due to this length encoding.

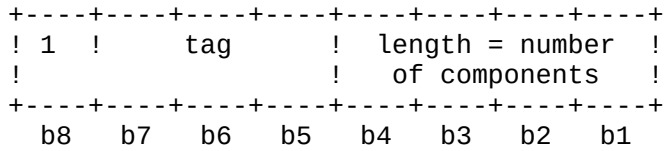


Figure 14. Coding of ID Info for a SEQUENCE

A structure may contain user data or further structures as components. Single components may be OPTIONAL, i.e. they may be omitted. In this case the ID Info is omitted too. A SEQUENCE shall be counted as a single component even if it contains several components.

EXAMPLE: The hexadecimal notation is used for the following example of encoding. The upper case letter X is used as a fill-in for unknown values, such as the length of the single components or the tag of the structure. A lower case letter x represents user data.

Syntax Description	Code	comment
Person [1] IMPLICIT SEQUENCE {	94	4 components
[0] Surname,	0X xx ...	tag 0
[1] First name,	1X xx ...	tag 1
[2] City,	2X xx ...	tag 2
[3] Street	3X xx ...	tag 3
}		

4.3.2.2 SEQUENCE OF

The SEQUENCE OF structure represents a succession of components. It is comparable with an array. The structure may contain one or more components. The components may be user data or structures.

The coding is as for the structure SEQUENCE. For the statement of the number of components the number of repetitions shall be taken into account. The tags of the Syntax Description shall be used for the components of SEQUENCE OF. The same tags may be coded several times in succession.

EXAMPLE:

```
employeedata [2] IMPLICIT SEQUENCE OF {
  [0] Person
  }
```

4.3.2.3 CHOICE

A CHOICE represents a selection from a set of predefined possibilities.

The components of a CHOICE construct shall have different tags to allow proper identification.

Instead of the CHOICE construct the actually selected component is encoded.

EXAMPLE:

```
Data ::= CHOICE {
  [0] Employeedata,
  [1] Clientdata,
  [2] Supplierdata
  }
```

5 Syntax Description

5.1 The FMS PDU

5.1.1 Description of the fixed PDU Part

The FMS PDU consists of a fixed part of 3 octets and a part of variable length. Not all PDUs need a part with variable length.

The fixed part consists of a first ID Info, which states the service class (e.g. Confirmed Request, Confirmed Response), an Invoke ID (1 octet, Data Type Integer8) and a second ID Info which identifies the PDU more precisely.

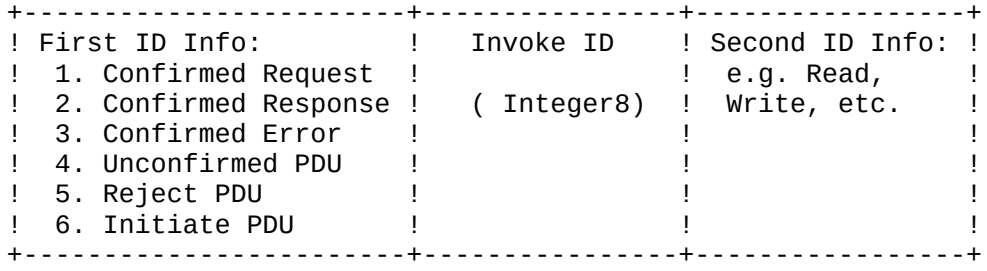


Figure 15. Field with Fixed Format

The syntax of the PDU is described in the formal description language ASN.1. The syntax of the fixed field is described in ASN.1 too. The special length saving FER (Fieldbus Encoding Rules) are used. The FER allow data fields inside the PDU, which have no explicit Identification Information (for the Invoke ID).

The fixed format for the first three octets requires an InvokeID even for PDUs that need no InvokeID (such as Unconfirmed-PDU, Reject-PDU, Initiate-PDU). Only the range from 0 to 6 may be used for the tag of the first ID Info. Otherwise an extension would be necessary, which would require one additional octet and would then be incompatible with a fixed format (fixed position of the InvokeID).

The fixed part of the PDU is defined in the following.

FMS PDU:

```
FMSpdu ::= CHOICE {
  confirmed-RequestPDU     [1] IMPLICIT Confirmed-RequestPDU,
  confirmed-ResponsePDU    [2] IMPLICIT Confirmed-ResponsePDU,
  confirmed-ErrorPDU       [3] IMPLICIT Confirmed-ErrorPDU,
  unconfirmed-PDU          [4] IMPLICIT Unconfirmed-PDU,
  reject-PDU               [5] IMPLICIT Reject-PDU,
  initiate-PDU             [6] IMPLICIT Initiate-PDU
  }

  Confirmed-RequestPDU ::= SEQUENCE {
    invokeID                    InvokeID,
    confirmedServiceRequest     ConfirmedServiceRequest
    }

  Confirmed-ResponsePDU ::= SEQUENCE {
    invokeID                    InvokeID,
    confirmedServiceResponse    ConfirmedServiceResponse
    }

  Confirmed-ErrorPDU ::= SEQUENCE {
    invokeID         InvokeID,
    serviceError     ServiceError
    }

  Unconfirmed-PDU ::= SEQUENCE {
    invokeID       InvokeID, -- Dummy
    unconfirmed     UnconfirmedService
    }

  Reject-PDU ::= SEQUENCE {
    invokeID        InvokeID, -- Dummy
    reject          Reject
    }

  Initiate-PDU ::= SEQUENCE {
    invokeID        InvokeID, -- Dummy
    initiatepdu     InitiatePDU
    }

InvokeID ::=  Integer8
```

5.1.2 ConfirmedServiceRequest

```
ConfirmedServiceRequest ::= CHOICE {
  status                           [0] IMPLICIT Status-Request,
  identify                         [1] IMPLICIT Identify-Request,
  read                             [2] IMPLICIT Read-Request,
  write                            [3] IMPLICIT Write-Request,
  getOD                            [4] IMPLICIT GetOD-Request,
  readWithType                     [5] IMPLICIT ReadWithType-Request,
  writeWithType                    [6] IMPLICIT WriteWithType-Request,
  defineVariableList               [7] IMPLICIT DefineVariableList-Request,
  deleteVariableList               [8] IMPLICIT DeleteVariableList-Request,
  initiateDownloadSequence         [9] IMPLICIT InitiateDownloadSequence-Request,
  downloadSegment                 [10] IMPLICIT DownloadSegment-Request,
  terminateDownloadSequence       [11] IMPLICIT TerminateDownloadSequence-Request,
  initiateuploadSequence          [12] IMPLICIT InitiateUploadSequence-Request,
  uploadSegment                   [13] IMPLICIT UploadSegment-Request,
  terminateUploadSequence         [14] IMPLICIT TerminateUploadSequence-Request,
  requestDomainDownload           [15] IMPLICIT RequestDomainDownload-Request,
  requestDomainUpload             [16] IMPLICIT RequestDomainUpload-Request,
  createProgramInvocation         [17] IMPLICIT CreateProgramInvocation-Request,
  deleteProgramInvocation         [18] IMPLICIT DeleteProgramInvocation-Request,
  start                           [19] IMPLICIT Start-Request,
  stop                            [20] IMPLICIT Stop-Request,
  resume                          [21] IMPLICIT Resume-Request,
  reset                           [22] IMPLICIT Reset-Request,
  kill                            [23] IMPLICIT Kill-Request,
  alterEventConditionMonitoring   [24] IMPLICIT
                                       AlterEventConditionMonitoring-Request,
  acknowledgeEventNotification    [25] IMPLICIT
                                       AcknowledgeEventNotification-Request,
  physRead                        [26] IMPLICIT PhysRead-Request,
  physWrite                       [27] IMPLICIT PhysWrite-Request,
  initiatePutOD                   [28] IMPLICIT InitiatePutOD-Request,
  putOD                           [29] IMPLICIT PutOD-Request,
  terminatePutOD                  [30] IMPLICIT TerminatePutOD-Request
  }
```

5.1.3 ConfirmedServiceResponse

```
ConfirmedServiceResponse ::= CHOICE {
  status                          [0] IMPLICIT Status-Response,
  identify                        [1] IMPLICIT Identify-Response,
  read                            [2] IMPLICIT Read-Response,
  write                           [3] IMPLICIT Write-Response,
  getOD                           [4] IMPLICIT GetOD-Response,
  readWithType                    [5] IMPLICIT ReadWithType-Response,
  writeWithType                   [6] IMPLICIT WriteWithType-Response,
  defineVariableList              [7] IMPLICIT DefineVariableList-Response,
  deleteVariableList              [8] IMPLICIT DeleteVariableList-Response,
  initiateDownloadSequence        [9] IMPLICIT InitiateDownloadSequence-Response,
  downloadSegment                [10] IMPLICIT DownloadSegment-Response,
  terminateDownloadSequence      [11] IMPLICIT
                                        TerminateDownloadSequence-Response,
  initiateuploadSequence         [12] IMPLICIT
                                         InitiateUploadSequence-Response,
  uploadSegment                  [13] IMPLICIT UploadSegment-Response,
  terminateUploadSequence        [14] IMPLICIT TerminateUploadSequence-Response,
  requestDomainDownload          [15] IMPLICIT RequestDomainDownload-Response,
  requestDomainUpload            [16] IMPLICIT RequestDomainUpload-Response,
  createProgramInvocation        [17] IMPLICIT CreateProgramInvocation-Response,
  deleteProgramInvocation        [18] IMPLICIT DeleteProgramInvocation-Response,
  start                          [19] IMPLICIT Start-Response,
  stop                           [20] IMPLICIT Stop-Response,
  resume                         [21] IMPLICIT Resume-Response,
  reset                          [22] IMPLICIT Reset-Response,
  kill                           [23] IMPLICIT Kill-Response,
  alterEventConditionMonitoring  [24] IMPLICIT
                                        AlterEventConditionMonitoring-Response,
  acknowledgeEventNotification   [25] IMPLICIT
                                         AcknowledgeEventNotification-Response,
  physRead                       [26] IMPLICIT PhysRead-Response,
  physWrite                      [27] IMPLICIT PhysWrite-Response,
  initiatePutOD                  [28] IMPLICIT InitiatePutOD-Response,
  putOD                          [29] IMPLICIT PutOD-Response,
  terminatePutOD                 [30] IMPLICIT TerminatePutOD-Response
  }
```

5.1.4 ServiceError

```
ServiceError ::= CHOICE {
  status                          [0] IMPLICIT Error-Type,
  identify                        [1] IMPLICIT Error-Type,
  read                            [2] IMPLICIT Error-Type,
  write                           [3] IMPLICIT Error-Type,
  getOD                           [4] IMPLICIT Error-Type,
  readWithType                    [5] IMPLICIT Error-Type,
  writeWithType                   [6] IMPLICIT Error-Type,
  defineVariableList              [7] IMPLICIT Error-Type,
  deleteVariableList              [8] IMPLICIT Error-Type,
  initiateDownloadSequence        [9] IMPLICIT Error-Type,
  downloadSegment                [10] IMPLICIT Error-Type,
  terminateDownloadSequence      [11] IMPLICIT Error-Type,
  initiateuploadSequence         [12] IMPLICIT Error-Type,
  uploadSegment                  [13] IMPLICIT Error-Type,
  terminateUploadSequence        [14] IMPLICIT Error-Type,
  requestDomainDownload          [15] IMPLICIT Error-Type,
  requestDomainUpload            [16] IMPLICIT Error-Type,
  createProgramInvocation        [17] IMPLICIT Error-Type,
  deleteProgramInvocation        [18] IMPLICIT Error-Type,
  start                          [19] IMPLICIT PI-Error-Type,
  stop                           [20] IMPLICIT PI-Error-Type,
  resume                         [21] IMPLICIT PI-Error-Type,
  reset                          [22] IMPLICIT PI-Error-Type,
  kill                           [23] IMPLICIT Error-Type,
  alterEventConditionMonitoring  [24] IMPLICIT Error-Type,
  acknowledgeEventNotification   [25] IMPLICIT Error-Type,
  physRead                       [26] IMPLICIT Error-Type,
  physWrite                      [27] IMPLICIT Error-Type,
  initiatePutOD                  [28] IMPLICIT Error-Type,
  putOD                          [29] IMPLICIT Error-Type,
  terminatePutOD                 [30] IMPLICIT OD-Error-Type
  }
```

5.1.4.1 Error Type

```
Error-Type ::= SEQUENCE {
  error-class               [0] IMPLICIT error-class,
  additional-code           [1] IMPLICIT Integer16 OPTIONAL,
  additional-description    [2] IMPLICIT VISIBLE STRING OPTIONAL
  }
```

5.1.4.2 PI Error Type

```
PI-Error-Type ::= SEQUENCE {
  error-class               [0] IMPLICIT Error-Class,
  additional-code           [1] IMPLICIT Integer16 OPTIONAL,
  additional-description    [2] IMPLICIT VISIBLE STRING OPTIONAL,
  pi-state                  [3] IMPLICIT ProgramInvocationState
  }
```

5.1.4.3 OD Error Type

```
ODErrorType ::= SEQUENCE {
  error-class               [0] IMPLICIT Error-Class,
  additional-code           [1] IMPLICIT Integer16 OPTIONAL,
  additional-description    [2] IMPLICIT VISIBLE STRING OPTIONAL,
  index                     [3] IMPLICIT Index
  }
```

5.1.4.4 Error Class

```
error-class ::=  SEQUENCE {
  CHOICE {
    vfd-state               [1] IMPLICIT  Integer8 {
      other                                (0)
      },
    application-reference   [2] IMPLICIT  Integer8 {
      other                                (0),
      application-unreachable              (1)
      },
    definition              [3] IMPLICIT  Integer8 {
      other                                (0),
      object-undefined                     (1),
      object-attributes-inconsistent       (2),
      name-already-exists                  (3)
      },
    resource                [4] IMPLICIT  Integer8 {
      other                                (0),
      memory-unavailable                   (1)
      },
    service                 [5] IMPLICIT  Integer8 {
      other                                (0),
      object-state-conflict                (1),
      pdu-size                             (2),
      object-constraint-conflict           (3),
      parameter-inconsistent               (4),
      illegal-parameter                    (5)
      },
    access                  [6] IMPLICIT  Integer8 {
      other                                (0),
      object-invalidated                   (1),
      hardware-fault                       (2),
      object-access-denied                 (3),
      invalid-address                      (4),
      object-attribute-inconsistent        (5),
      object-access-unsupported            (6),
      object-non-existent                  (7),
      type-conflict                        (8),
      named-access-unsupported             (9)
      },
    od                      [7] IMPLICIT  Integer8 {
      other                                (0),
      name-length-overflow                 (1),
      od-overflow                          (2),
      od-write-protected                   (3),
      extension-length-overflow            (4),
      od-description-length-overflow       (5),
      operational-problem                  (6)
      },
    other                   [8] IMPLICIT  Integer8 {
      other                                (0)
    }}}
```

5.1.5 Unconfirmed PDUs

```
UnconfirmedService ::= CHOICE {
  informationReport           [0] IMPLICIT InformationReport,
  unsolicitedStatus           [1] IMPLICIT UnsolicitedStatus,
  eventNotification           [2] IMPLICIT EventNotification,
  informationReportWithType   [3] IMPLICIT InformationReportWithType,
  eventNotificationWithType   [4] IMPLICIT EventNotificationWithType
  }
```

5.1.6 Reject

```
Reject ::= [0] IMPLICIT SEQUENCE {
  original-invokeId    [0] IMPLICIT  Integer8,
  reject-code          [1] IMPLICIT  Integer8 {
    other                   (0),
    PDU-Size                (5)
    }}
```

5.1.7 Initiate PDUs

```
InitiatePDU ::= CHOICE {
  init-request     [0] IMPLICIT Initiate-RequestPDU,
  init-response    [1] IMPLICIT Initiate-ResponsePDU,
  init-error       [2] IMPLICIT Initiate-ErrorPDU
  }
```

5.1.8 General Substitutions

```
Index         ::= Unsigned16

Name          ::= VISIBLE STRING

Subindex      ::= Unsigned8

Local-Address ::= Unsigned32

Length        ::= Unsigned8

MoreFollows   ::= BOOLEAN

Data          ::= PACKED


Typedescription ::= SEQUENCE OF CHOICE {

simple     [1]  IMPLICIT PACKED,
                -- Octet1: Hibyte Data Type Index
                -- Octet2: Lobyte Data Type Index
                -- Octet3: Length

array      [2]  IMPLICIT PACKED,
                -- Octet1: Hibyte Data Type Index
                -- Octet2: Lobyte Data Type Index
                -- Octet3: Length
                -- Octet4: Number Of Elements

structure  [3]  IMPLICIT PACKED
                -- Octet 1    : Hibyte Data Type Index (Element 1)
                -- Octet 2    : Lobyte Data Type Index (Element 1)
                -- Octet 3    : Length                 (Element 1)
                -- Octet 3n-2 : Hibyte Data Type Index (Element n)
                -- Octet 3n-1 : Lobyte Data Type Index (Element n)
                -- Octet 3n-0 : Length                 (Element n)
  }


Integer8  ::= INTEGER  --  8 Bit Integer
Integer16 ::= INTEGER  -- 16 Bit Integer
Integer32 ::= INTEGER  -- 32 Bit Integer

Unsigned8  ::= UNSIGNED  --  8 Bit Unsigned
Unsigned16 ::= UNSIGNED  -- 16 Bit Unsigned
Unsigned32 ::= UNSIGNED  -- 32 Bit Unsigned
```

5.2 VFD Support

5.2.1 Status

```
Status-Request ::= NULL

Status-Response ::= SEQUENCE {
  logical-status         [0] IMPLICIT Unsigned8 {
    state-changes-allowed      (0),
    limited-services-permitted (2),

    od-loading-non-interacting (4),
    od-loading-interacting     (5)
    },
  physical-status        [1] IMPLICIT Unsigned8 {
    operational                (0),
    partially-operational      (1),
    inoperable                 (2),
    needs-commissioning        (3)
    },
  local-detail           [2] IMPLICIT BIT STRING OPTIONAL
                                    -- Length 24 Bit
  }
```

5.2.2 Identify

```
Identify-Request ::= NULL

Identify-Response ::= SEQUENCE {
  vendor-name    [0] IMPLICIT VISIBLE STRING,
  model-name     [1] IMPLICIT VISIBLE STRING,
  revision       [2] IMPLICIT VISIBLE STRING
  }
```

5.2.3 UnsolicitedStatus

```
UnsolicitedStatus ::= Status-Response
```

5.3 OD Management

5.3.1 Object Description and OD Description

```
Objectdescription ::= PACKED
```

5.3.2 GetOD

```
GetOD-Request ::= SEQUENCE {
  all-attributes        [0] IMPLICIT BOOLEAN,
  access-specification  CHOICE {
    index               [1] IMPLICIT Index,
    variable-name       [2] IMPLICIT Name,
    variable-list-name  [3] IMPLICIT Name,
    domain-name         [4] IMPLICIT Name,
    pi-name             [5] IMPLICIT Name,
    event-name          [6] IMPLICIT Name,
    startindex          [7] IMPLICIT Index
    }
  }


GetOD-Response ::= SEQUENCE {
  list-of-objectdescription  [0] IMPLICIT SEQUENCE OF {
    objectdescription    [0] IMPLICIT Objectdescription
    },
  more-follows               [1] IMPLICIT MoreFollows
  }
```

5.3.3 InitiatePutOD

```
InitiatePutOD-Request ::= Consequence

Consequence ::=  Integer8 {
  od-loading-non-interacting   (0),
  od-appending-interacting     (1),
  od-fresh-loading-interacting (2)
  }

InitiatePutOD-Response ::= NULL
```

5.3.4 PutOD

```
PutOD-Request ::= SEQUENCE OF {
  objectdescription  [0] IMPLICIT Objectdescription
  }

PutOD-Response ::= NULL
```

5.3.5 TerminatePutOD

```
TerminatePutOD-Request ::= NULL

TerminatePutOD-Response ::= NULL
```

5.4 Context Management

5.4.1 AccessControl

```
Accesscontrol ::= BIT STRING {
  Password_Bit1 (7),
  Password_Bit2 (6),
  Password_Bit3 (5),
  Password_Bit4 (4),
  Password_Bit5 (3),
  Password_Bit6 (2),
  Password_Bit7 (1),
  Password_Bit8 (0),
  Access_Groups-1 (15),
  Access_Groups-2 (14),
  Access_Groups-3 (13),
  Access_Groups-4 (12),
  Access_Groups-5 (11),
  Access_Groups-6 (10),
  Access_Groups-7 (9),
  Access_Groups-8 (8)
  }
  -- The Password (Unsigned8) is encoded as a bit string. The
  -- mapping of the Data Type unsigned8 to the specified bit
  -- number is defined according to unsigned definition.
```

5.4.2 Initiate

```
Initiate-RequestPDU ::= SEQUENCE {
  version-od_calling                  [0] IMPLICIT Integer16,
  profile-number_calling              [1] IMPLICIT OCTET STRING,
  access-protection-supported_calling [2] IMPLICIT BOOLEAN,
  password_and_access-groups_calling  [3] IMPLICIT Accesscontrol,
  max-pdu-sending-high-prio_calling   [4] IMPLICIT Unsigned8,
  max-pdu-sending-low-prio_calling    [5] IMPLICIT Unsigned8,
  max-pdu-receiving-high-prio_calling [6] IMPLICIT Unsigned8,
  max-pdu-receiving-low-prio_calling  [7] IMPLICIT Unsigned8,
  fms-features-supported_calling      [8] IMPLICIT BIT STRING
                                           -- Encoding according
                                           -- to context management definition
  }


Initiate-ResponsePDU ::= SEQUENCE {
  version-od_called                  [0] IMPLICIT Integer16,
  profile-number_called              [1] IMPLICIT OCTET STRING,
  access-protection-supported_called [2] IMPLICIT BOOLEAN,
  password_and_access-groups_called  [3] IMPLICIT Accesscontrol
  }
```

```
Initiate-ErrorPDU ::= SEQUENCE {
  error-code                          [0] IMPLICIT  Integer8 {
    other                        (0),
    max-pdu-size-insufficient    (1),
    feature-not-supported        (2),
    version-od-incompatible      (3),
    user-initiate-denied         (4),
    password-error               (5),
    profile-number-incompatible (6)
    },
  max-pdu-sending-high-prio-called   [1] IMPLICIT Unsigned8,
  max-pdu-sending-low-prio-called    [2] IMPLICIT Unsigned8,
  max-pdu-receiving-high-prio-called [3] IMPLICIT Unsigned8,
  max-pdu-receiving-low-prio-called  [4] IMPLICIT Unsigned8,
  fms-features-supported-called      [5] IMPLICIT BIT STRING
                                       -- Encoding according
                                       -- to context management definition
 }
```

5.5 Domain Management

5.5.1 The Domain Object

```
Domain-State ::= Unsigned8 {
  Existent      (1),
  Loading       (2),
  Incomplete    (3),
  Complete      (4),
  Ready         (5),
  In-Use        (6)
  }

Upload-State ::= Unsigned8 {
  Non-existent  (0),
  Uploading     (1),
  Uploaded      (2)
  }
```

5.5.2 InitiateDownloadSequence

```
InitiateDownloadSequence-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index         [0] IMPLICIT Index,
    domain-name   [1] IMPLICIT Name
    }
  }

InitiateDownloadSequence-Response ::= NULL
```

5.5.3 DownloadSegment

```
DownloadSegment-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index         [0] IMPLICIT Index,
    domain-name   [1] IMPLICIT Name
    }
  }

DownloadSegment-Response ::= SEQUENCE {
  load-data       [0] IMPLICIT OCTET STRING,
  more-follows    [1] IMPLICIT MoreFollows
  }
```

5.5.4 TerminateDownloadSequence

```
TerminateDownloadSequence-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index         [0] IMPLICIT Index,
    domain-name   [1] IMPLICIT Name
    },
  final-result    [2] IMPLICIT BOOLEAN
  }

TerminateDownloadSequence-Response ::= NULL
```

5.5.5 InitiateUploadSequence

```
InitiateUploadSequence-Request ::= SEQUENCE {
  access-specification   CHOICE {
    index         [0] IMPLICIT Index,
    domain-name   [1] IMPLICIT Name
    }
  }

InitiateUploadSequence-Response ::= NULL
```

5.5.6 UploadSegment

```
UploadSegment-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index         [0] IMPLICIT Index,
    domain-name   [1] IMPLICIT Name
    }
  }

UploadSegment-Response ::= SEQUENCE {
  load-data       [0] IMPLICIT OCTET STRING,
  more-follows    [1] IMPLICIT MoreFollows
  }
```

5.5.7 TerminateUploadSequence

```
TerminateUploadSequence-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index         [0] IMPLICIT Index,
    domain-name   [1] IMPLICIT Name
    }
  }

TerminateUploadSequence-Response ::= NULL
```

5.5.8 RequestDomainDownload

```
RequestDomainDownload-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index                [0] IMPLICIT Index,
    domain-name          [1] IMPLICIT Name
    },
  additionalInformation  [2] IMPLICIT VISIBLE STRING OPTIONAL
  }

RequestDomainDownload-Response ::= NULL
```

5.5.9 RequestDomainUpload

```
RequestDomainUpload-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index                [0] IMPLICIT Index,
    domain-name          [1] IMPLICIT Name
    },
  additionalInformation  [2] IMPLICIT VISIBLE STRING OPTIONAL
  }

RequestDomainUpload-Response ::= NULL
```

5.6 Program Invocation Management

5.6.1 ProgramInvocationState

```
ProgramInvocationState ::=  Integer8 {
  non-existent  (0),  -- NON-EXISTENT
  unrunnable    (1),  -- UNRUNNABLE
  idle          (2),  -- IDLE
  running       (3),  -- RUNNING
  stopped       (4),  -- STOPPED
  starting      (5),  -- STARTING
  stopping      (6),  -- STOPPING
  resuming      (7),  -- RESUMING
  resetting     (8)   -- RESETTING
  }

PI-Access-Protection ::= BIT STRING {
  S    (23),
  H    (22),
  D    (21),
  Sg   (19),
  Hg   (18),
  Dg   (17),
  Sa   (31),
  Ha   (30),
  Da   (29),
  Password_Bit1 (7),
  Password_Bit2 (6),
  Password_Bit3 (5),
  Password_Bit4 (4),
  Password_Bit5 (3),
  Password_Bit6 (2),
  Password_Bit7 (1),
  Password_Bit8 (0),
  Access_Groups-1 (15),
  Access_Groups-2 (14),
  Access_Groups-3 (13),
  Access_Groups-4 (12),
  Access_Groups-5 (11),
  Access_Groups-6 (10),
  Access_Groups-7 (9),
  Access_Groups-8 (8)
  }
  -- The Password (unsigned8) is encoded as a bit string. The
  -- mapping of the Data Type unsigned8 to the specified bit
  -- number is defined according to the unsigned definition.
```

5.6.2 CreateProgramInvocation

```
CreateProgramInvocation-Request ::= SEQUENCE {
  listOfDomains     [0] IMPLICIT SEQUENCE OF CHOICE {
    index         [0] IMPLICIT Index,
    domain-name   [1] IMPLICIT Name
    },
  access-protection [1] IMPLICIT PI-Access-Protection,
  pi-name           [2] IMPLICIT Name OPTIONAL,
  extension         [3] IMPLICIT PACKED OPTIONAL,
  reusable          [4] IMPLICIT BOOLEAN
  }

CreateProgramInvocation-Response ::= Index
```

5.6.3 DeleteProgramInvocation

```
DeleteProgramInvocation-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index         [0] IMPLICIT Index,
    pi-name       [1] IMPLICIT Name
    }
  }

DeleteProgramInvocation-Response ::= NULL
```

5.6.4 Start

```
Start-Request ::= SEQUENCE {
  access-specification   CHOICE {
    index             [0] IMPLICIT Index,
    pi-name           [1] IMPLICIT Name
    },
  execution-argument  [2] IMPLICIT OCTET STRING OPTIONAL
  }

Start-Response ::= NULL
```

5.6.5 Stop

```
Stop-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index         [0] IMPLICIT Index,
    pi-name       [1] IMPLICIT Name
    }
  }

Stop-Response ::= NULL
```

5.6.6 Resume

```
Resume-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index             [0] IMPLICIT Index,
    pi-name           [1] IMPLICIT Name
    },
  execution-argument  [2] IMPLICIT OCTET STRING OPTIONAL
  }

Resume-Response ::= NULL
```

5.6.7 Reset

```
Reset-Request ::= SEQUENCE {
  access-specification   CHOICE {
    index         [0] IMPLICIT Index,
    pi-name       [1] IMPLICIT Name
    }
  }

Reset-Response ::= NULL
```

5.6.8 Kill

```
Kill-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index         [0] IMPLICIT Index,
    pi-name       [1] IMPLICIT Name
    }
  }

Kill-Response ::= NULL
```

5.7 Variable Access

5.7.1 VariableListAccessProtection

```
Variable-List-Access-Protection ::= BIT STRING {
  R    (23),
  W    (22),
  D    (21),
  Rg   (19),
  Wg   (18),
  Dg   (17),
  Ra   (31),
  Wa   (30),
  Da   (29),
  Password_Bit1 (7),
  Password_Bit2 (6),
  Password_Bit3 (5),
  Password_Bit4 (4),
  Password_Bit5 (3),
  Password_Bit6 (2),
  Password_Bit7 (1),
  Password_Bit8 (0),
  Access_Groups-1 (15),
  Access_Groups-2 (14),
  Access_Groups-3 (13),
  Access_Groups-4 (12),
  Access_Groups-5 (11),
  Access_Groups-6 (10),
  Access_Groups-7 (9),
  Access_Groups-8 (8)
  }
  -- The Password (unsigned8) is encoded as a bit string. The
  -- mapping of the Data Type unsigned8 to the specified bit
  -- number is defined according to the unsigned definition.
```

5.7.2 Read

```
Read-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index               [0] IMPLICIT Index,
    variable-name       [1] IMPLICIT Name,
    variable-list-name  [2] IMPLICIT Name
    },
  subindex              [3] IMPLICIT Subindex OPTIONAL
  }

Read-Response ::= Data
```

5.7.3 Write

```
Write-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index               [0] IMPLICIT Index,
    variable-name       [1] IMPLICIT Name,
    variable-list-name  [2] IMPLICIT Name
    },
  subindex              [3] IMPLICIT Subindex OPTIONAL,
  data                  [4] IMPLICIT Data
  }

Write-Response ::= NULL
```

5.7.4 DefineVariableList

```
DefineVariableList-Request ::= SEQUENCE {
  listOfVariables     [0] IMPLICIT SEQUENCE OF CHOICE {
    index               [0] IMPLICIT Index,
    variable-name       [1] IMPLICIT Name
    },
  access-protection   [1] IMPLICIT
                          Variable-List-Access-Protection,
  variable-list-name  [2] IMPLICIT Name OPTIONAL,
  extension           [3] IMPLICIT PACKED OPTIONAL
  }

DefineVariableList-Response ::= Index
```

5.7.5 DeleteVariableList

```
DeleteVariableList-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index               [0] IMPLICIT Index,
    variable-list-name  [1] IMPLICIT Name
    }
  }

DeleteVariableList-Response ::= NULL
```

5.7.6 PhysRead

```
PhysRead-Request ::= SEQUENCE {
  local-address       [0] IMPLICIT Local-Address,
  length              [1] IMPLICIT Length
  }

PhysRead-Response ::= Data
```

5.7.7 PhysWrite

```
PhysWrite-Request ::= SEQUENCE {
  local-address       [0] IMPLICIT Local-Address,
  data                [1] IMPLICIT Data
  }

PhysWrite-Response ::= NULL
```

5.7.8 InformationReport

```
InformationReport ::= SEQUENCE {
  access-specification   CHOICE {
    index               [0] IMPLICIT Index,
    variable-name       [1] IMPLICIT Name,
    variable-list-name  [2] IMPLICIT Name
    },
  subindex              [3] IMPLICIT Subindex OPTIONAL,
  data                  [4] IMPLICIT Data
  }
```

5.7.9 ReadWithType

```
ReadWithType-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index              [0] IMPLICIT Index,
    variable-name      [1] IMPLICIT Name,
    variable-list-name [2] IMPLICIT Name
    },
  subindex             [3] IMPLICIT Subindex OPTIONAL
  }

ReadWithType-Response ::= SEQUENCE {
  typedescription [0] IMPLICIT Typedescription,
  data            [1] IMPLICIT Data
  }
```

5.7.10 WriteWithType

```
WriteWithType-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index                [0] IMPLICIT Index,
    variable-name        [1] IMPLICIT Name,
    variable-list-name   [2] IMPLICIT Name
    },
  subindex               [3] IMPLICIT Subindex OPTIONAL,
  typedescription        [4] IMPLICIT Typedescription,
  data                   [5] IMPLICIT Data
  }

WriteWithType-Response ::= NULL
```

5.7.11 InformationReportWithType

```
InformationReportWithType ::= SEQUENCE {
  access-specification  CHOICE {
    index              [0] IMPLICIT Index,
    variable-name      [1] IMPLICIT Name,
    variable-list-name [2] IMPLICIT Name
    },
  subindex             [3] IMPLICIT Subindex OPTIONAL,
  typedescription      [4] IMPLICIT Typedescription,
  data                 [5] IMPLICIT Data
  }
```

5.8 Event Management

5.8.1 AlterEventConditionMonitoring

```
AlterEventConditionMonitoring-Request ::= SEQUENCE {
  access-specification     CHOICE {
    index               [0] IMPLICIT Index,
    event-name          [1] IMPLICIT Name
    },
  enabled               [2] IMPLICIT BOOLEAN
  }

AlterEventConditionMonitoring-Response ::= NULL
```

5.8.2 AcknowledgeEventNotification

```
AcknowledgeEventNotification-Request ::= SEQUENCE {
  access-specification  CHOICE {
    index               [0] IMPLICIT Index,
    event-name          [1] IMPLICIT Name
    },
  eventNumber           [2] IMPLICIT Unsigned8
  }

AcknowledgeEventNotification-Response ::= NULL
```

5.8.3 EventNotification

```
EventNotification ::= SEQUENCE {
  access-specification  CHOICE {
    index               [0] IMPLICIT Index,
    event-name          [1] IMPLICIT Name
    },
  eventNumber           [2] IMPLICIT Unsigned8,
  eventData             [3] IMPLICIT Data OPTIONAL
  }
```

5.8.4 EventNotificationWithType

```
EventNotificationWithType ::= SEQUENCE {
  access-specification  CHOICE {
    index               [0] IMPLICIT Index,
    event-name          [1] IMPLICIT Name
    },
  eventNumber           [2] IMPLICIT Unsigned8,
  typedescription       [3] IMPLICIT Typedescription OPTIONAL,
  eventData             [4] IMPLICIT Data OPTIONAL
  }
```

5.9 Detailed Coding Examples

EXAMPLE: Knowledge of the syntax description of the PDUs is required to produce encoding examples. The example encodings are written in hexadecimal notation, whereby XX shall be replaced by user data.

READ-REQUEST_PDU:

The READ-REQUEST_PDU is a Confirmed-RequestPDU. A Confirmed-RequestPDU has the tag 1 in the FMS PDU CHOICE. It is a SEQUENCE consisting of 2 components (P/C flag =1, tag=1, length=2).

The first component is the InvokeID. Data Type and length of this component are known implicitly (universal tag - Integer8). Therefore the ID Info is omitted, i.e. there is only user data here.

The second component is a CHOICE (ConfirmedServiceRequest), from which the Read-Request with tag=2 is selected. This component is again a SEQUENCE. It consists of only one component - the index.

The Index has the Type unsigned16.

The Subindex is OPTIONAL and is omitted in this case.

```
Encoding    PDU
92          confirmed-RequestPDU [1] SEQUENCE {
XX            InvokeID,
A1            confirmedServiceRequest [2] SEQUENCE {
02 XX XX        [0] IMPLICIT Index
              }
            }
```

The READ-REQUEST_PDU with a Subindex is as follows:

The second SEQUENCE has 2 components and there is a Subindex with tag=3.

```
Encoding    PDU
92          confirmed-RequestPDU [1] SEQUENCE {
XX            InvokeID,
A2            confirmedServiceRequest [2] SEQUENCE {
02 XX XX        [0] IMPLICIT Index
31 XX           [3] IMPLICIT Subindex
              }
            }
```

The whole READ-REQUEST_PDU without Subindex has a length of 6 octets. Three of these octets contain user data.

Read PDUs:

```
Read-Request(InvID 8, Index 3)                92 08 A1 02 00 03
Read-Response(InvID 8, Integer16 1234 hex)    A2 08 22 12 34

Read-Request(InvID 5, Index 32)               92 05 A1 02 00 20
Read-Response(InvID 5, Boolean true)          A2 05 21 FF
```

PhysRead PDUs:

```
PhysRead-Request(InvID 6, Address 12345678 hex, 4 octets)
                                              92 06 F2 1A 04 12 34 56 78 11 04
PhysRead-Response(InvID 6, 47110815 hex)      A2 06 74 1A 47 11 08 15
```

GetOD PDUs:

```
GetOD-Request(InvID 7, All-Attributes FALSE, Index 27 )
                                        92 07 C2 01 00 72 07 00 1B
GetOD-Response(InvID 7, List of 2 object descriptions
                        object description:
                             Index 27,
                             Object Code Simple Variable,
                             Data Type Index  Integer8,
                             Length 1,
                        object description:
                             Index 28,
                             Object Code Simple Variable,
                             Data Type Index Octet String,
                             Length 5,
                        more-follows True )
                                A2 07 C2 82 06 00 1B 07 00 02 01
                                            06 00 1C 07 00 0A 05
                                            11 FF
```

6 Lower Layer Interface (LLI)

6.1 General

The Lower Layer Interface (LLI) represents the interface between the Fieldbus Message Specification (FMS) and the Layer 2 (FDL) with a part of the Layer 2 management (FMA1/2). In the same way, the LLI represents the interface between the Fieldbus Management Layer 7 (FMA7) and FDL, with FMA1/2 for remote management functions. In the following text, FMS and FMA7 are called LLI users.The definitions of the LLI are optimized for the FDL and the FMA1/2. Moreover the LLI provides an interface to the Layer 7 management (FMA7) for local management functions.

6.1.1 Tasks of the Lower Layer Interface (LLI)

- The LLI simulates the necessary Layer 3 to 6 functionality.
- The LLI provides a service interface to the LLI User independent of FDL.
- The LLI maps the FMS and FMA7 services onto the FDL services. The necessary FMA1/2 services are considered thereby.

6.1.2 Use of FDL Services and FMA1/2 Services

The LLI uses the following FDL services for the mapping of the LLI services onto the FDL services:

- Send Data with No Acknowledge (SDN)
- Send Data with Acknowledge (SDA)
- Send and Request Data with Reply (SRD)
- Cyclic Send and Request Data with Reply (CSRD)

At the active station (master) the services are processed as initiator, responder or both, at the passive station (slave) as responder. The following FMA1/2 services are used to set the parameters of the service access points (LSAPs):

- SAP Activate FMA1/2
- RSAP Activate FMA1/2
- SAP Deactivate FMA1/2

These FMA1/2 services are marked as optional in the PROFIBUS specification. They shall, however, be implemented to realize the functionality of the LLI; i.e. they are mandatory for LLI.

6.2 LLI Model

Between two application processes one or more communication relationships may exist. Communication end points are uniquely assigned to each communication relationship (see chapter 2). All communication relationships shall be configured in the Communication Reference List (CRL) independent of the time of use and shall be addressed with local communication references (CREF).

The configuration may either statically assign a communication end-point to a communication partner, or may allow the access to it for all partners (in the following called "open communication end-point"). In case of static assignment, no other communication partner has access to the communication end-point.

A communication relationship may either be connection-oriented or connectionless.

For a connection-oriented communication relationship a logical connection is established between two communication end-points of different communication partners (also called "one-to-one" communication relationship).

The LLI distinguishes three phases thereby:

- Connection establishment phase
- Data transfer phase
- Connection release phase

In the data transfer phase the communication end-point used shall be statically assigned to the end-point of the partner. For the open communication end-point this assignment is accomplished during the connection establishment phase.

Furthermore, the LLI distinguishes for connection oriented communication relationships between master-slave (M-S) and master-master (M-M). A master corresponds to an active station and a slave to a passive station according to the PROFIBUS Data Link Layer specification.

For a master-slave communication relationship logical connections for

- cyclic data transfer with no slave initiative or
- cyclic data transfer with slave initiative or
- acyclic data transfer with no slave initiative or
- acyclic data transfer with slave initiative

may exist.

A logical connection with slave initiative means that the slave may issue a unconfirmed service request to the master.

For the master-master communication relationship only the logical connection for

- acyclic data transfer

may exist.

If a connection for cyclic data transfer is to be configured between two masters, then one of the two masters shall emulate the slave for this connection. This connection is valid as a masterslave communication relationship. It is called a connection for cyclic data transfer with or with no slave initiative.

A connectionless communication relationship is either used for multicast data transfer (also called "one-to-many" communication relationship), or for broadcast data transfer (also called "one-to-all" communication relationship). These communication relationships are always in the data transfer phase and are not distinguished further.

6.2.1 LLI Addressing

An application process shall address its communication relationships with the help of communication references (CREF). Each CREF is assigned to exactly one local Link Service Access Point (local LSAP) of the Layer 2 (see PROFIBUS Data Link Layer, SSAP). On the other hand, some appointed local LSAPs may be assigned to multiple CREFs. Furthermore each CREF is assigned to exactly one LLI Service Access Point (LLI SAP). The assignment between CREF, the local LSAP and the LLI SAP is configured in the Communication Reference List (CRL) specifically to each communication relationship.

The CRL also contains the assignment of the CREF to:

- the remote address (see PROFIBUS Data Link Layer, rem_add) and
- the related remote LSAP (see PROFIBUS Data Link Layer, DSAP)

The LLI provides two service access points at the interface to the LLI user. The FMS shall use for all services the LLI SAP 0. The FMA7 shall use for all remote services the LLI SAP 1.

The LLI tests for every service request at the LLI user - LLI interface whether the service request may be assigned to a Layer 2 address (remote address, remote LSAP) which is configured in the CRL (see LLI CRL definition below). If the service request could be assigned to a Layer 2 address, the LLI enters the LLI SAP into the LLI PDU and delivers it to Layer 2 for transmission. Otherwise a fatal error has occurred. In this case the LLI ignores the service request and issues an error indication to the FMA7.

Upon receipt of a LLI PDU, the LLI tests whether the PDU could be assigned to a communication relationship which is registered in the CRL. If the PDU could be assigned to a CREF, the LLI delivers the received PDU to the LLI user at the assigned LLI SAP. PDUs which could not be assigned to a CREF are ignored by the LLI (see LLI CRL definition below).

To ensure a uniform access to all PROFIBUS stations by configuration or diagnostic devices, every station which supports the remote management services as responder, shall reserve the LSAP 1 and the CREF 1 for the default management connection (see FMA7). All other stations which do not support remote management, shall not use the LSAP1 and the CREF 1.

These assignments and definitions are shown in the following figure.

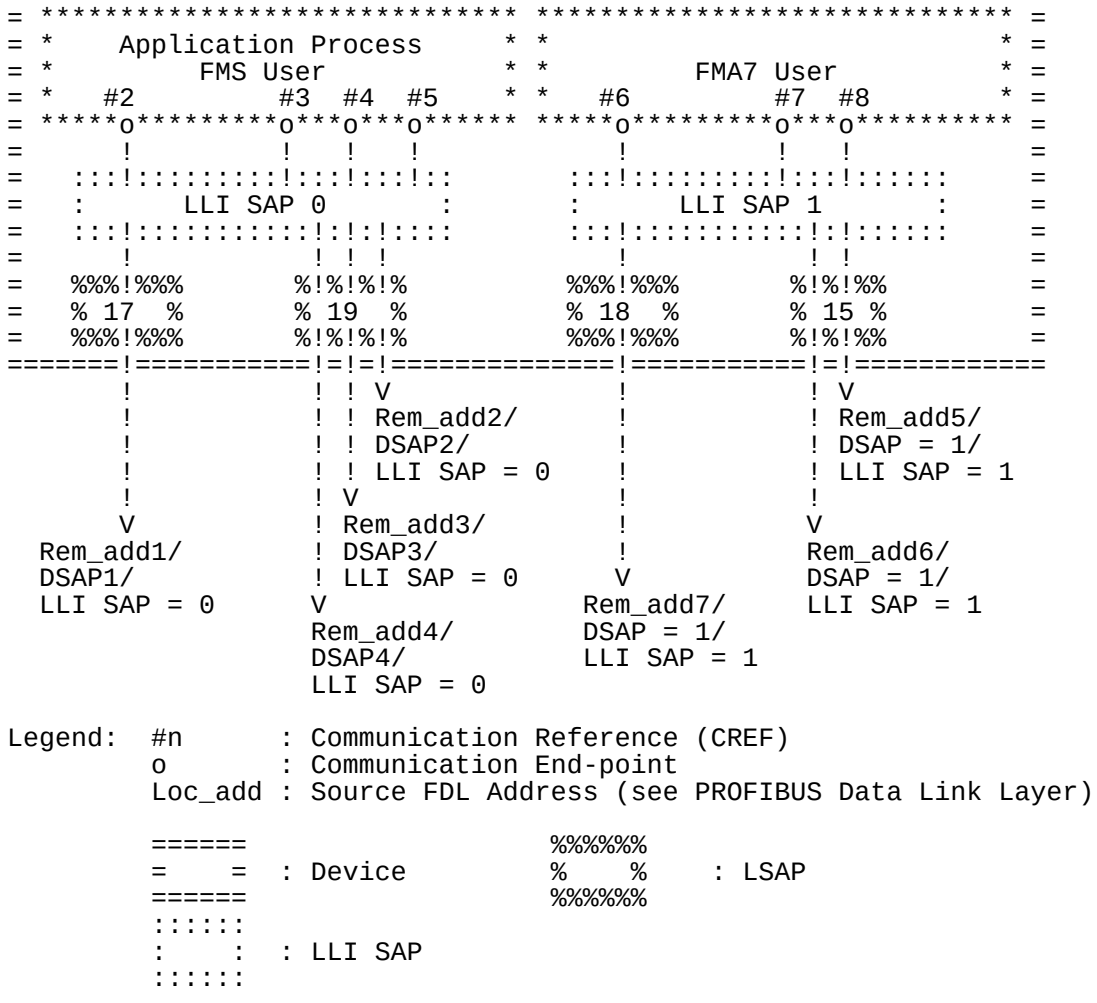


Figure 16. Addressing of Communication Relationships

6.2.2 Communication Relationships

The services of FMS and FMA7 are executed for connection-oriented communication relationships over logical connections. The connections may be distinguished from each other by their connection qualities (control mechanisms, resources etc.). Different requirements apply for master-slave communication relationships as compared to master-master communication relationships.

It is assumed for the devices at the slave side of master-slave communication relationships that they have only slave behaviour relative to the FMA7 and the confirmed FMS services. These devices may be client or server for unconfirmed FMS services.

Sensors usually need a fast and cyclic data exchange into a process image data memory at the master side. Actors often need to be actualized cyclically in order that a failure of the communication relationship to the master may be recognized (fail-safe, redundancy control etc.). The logical connection for cyclic data transfer with no slave initiative is appropriate for these applications.

Simple input / output devices, bar code readers, simple controllers or similar devices perform acyclic data transfer when the application process at the master works in spontaneous operation (loading of parameters, etc), or when the frequency of the data exchange is low (every 100 seconds, every 24 hours, process start-up, etc.). The logical connection for acyclic data transfer with no slave initiative is appropriate for these applications.

To indicate device errors and to notify process alarms, a priority controlled data transfer is necessary for intelligent slave devices with the initiative at the slave side. The logical connections for cyclic or acyclic data transfer with slave initiative are appropriate for these applications.

Master-master communication relationships allow parallel and mutual data exchange with priority controlled transmission. Thereby they may have client or server behaviour relative to the functionality.

Even between two master devices a cyclic data exchange may be necessary. The logical connection for cyclic data transfer is appropriate for these applications. Thereby one of the masters (the server of the confirmed services) shall behave like a slave, i.e. it is a master-slave communication relationship.

The unconfirmed services of the LLI user may also be executed on connectionless communication relationships (without logical connection). Broadcast or multicast communication relationships are appropriate for synchronization of stations, snapshots and exchange of global data to several or to all stations.

EXAMPLE: The characteristic communication relationships are illustrated for example in the following five figures and named " cyclic with / with no slave initiative, acyclic with / with no slave initiative, broadcast and multicast". Experience shows that devices for automatic control and systems in the field use these communication relationships side by side.

Notation in the following figures:

```
%   : Link Service Access Point (LSAP)
%%  : Poll List LSAP (see structure of LLI CRL definition below)
%%% : Global Link Service Access Point (LSAP 63, see connectionless
      CRL definition below)
```

a) 1-Master <--> n-Slaves

An examples of the communication relationships of one master to several slaves (sensors, actors and regulators) is shown in following figure.

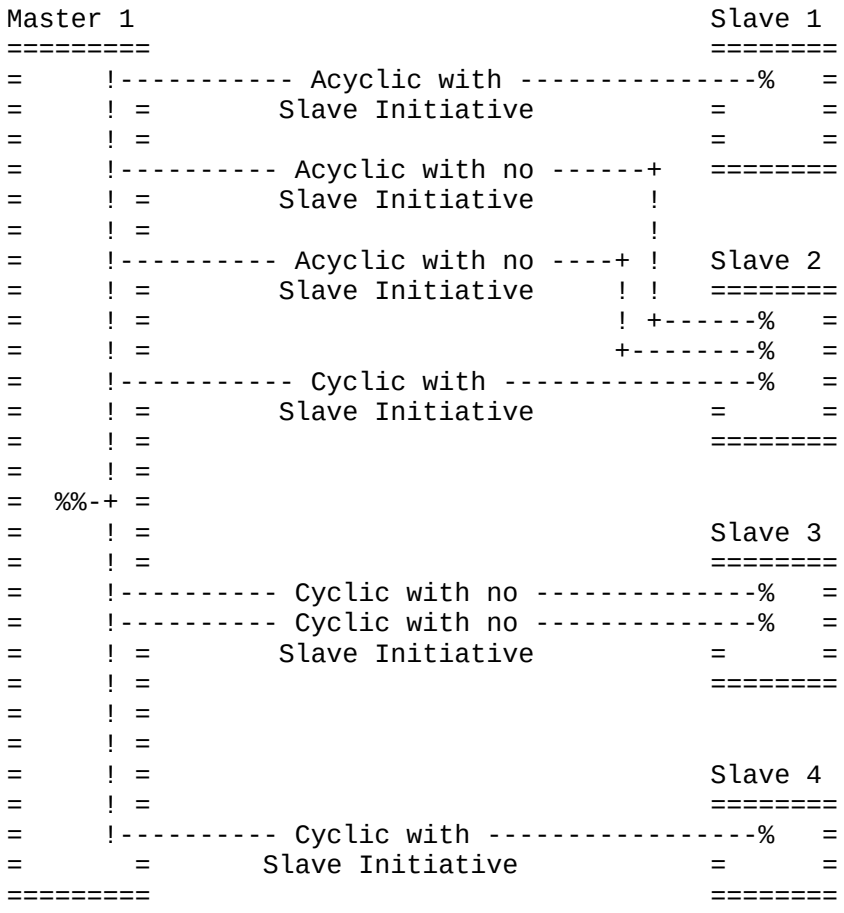


Figure 17. Communication Relationships between 1 Master and n Slaves

b) n-Master <--> 1-Slave

An example of the communication relationships of several masters to one slave is shown in the folowing figure. Thereby master 1 is a diagnostic or configuration station, master 2 as an intelligent terminal and master 3 is a process master.

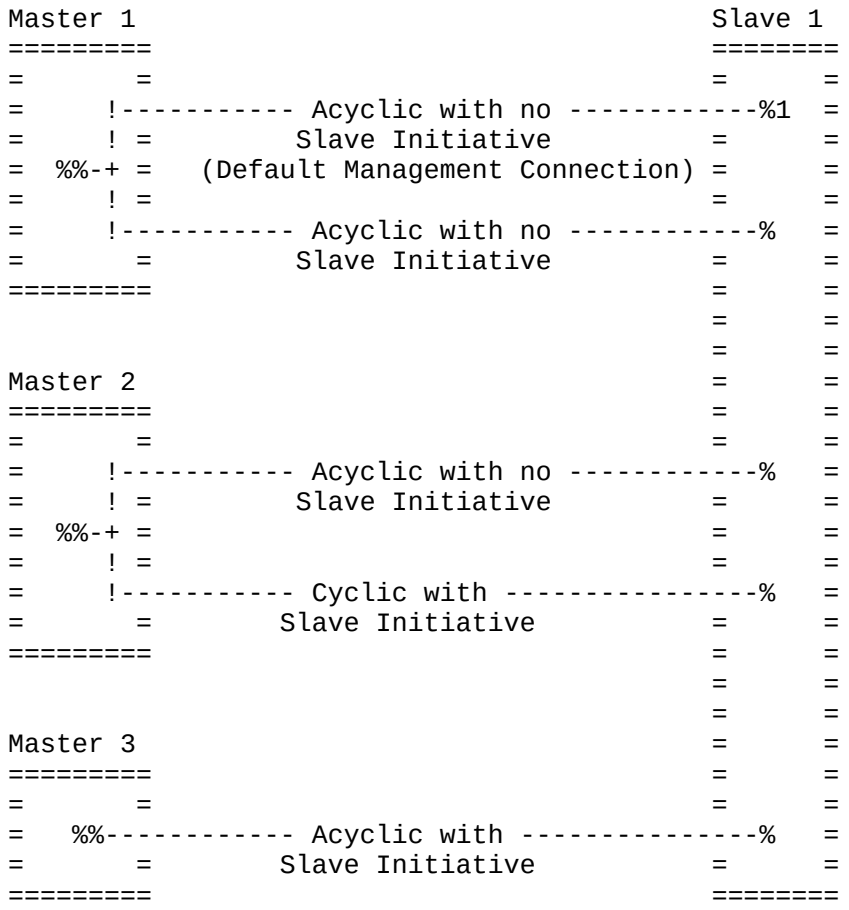


Figure 18. Communication Relationships between n Masters and 1 Slave

c) n-Master <--> n-Slaves

An example of the communication relationships of several masters to several slaves is shown in the following figure.

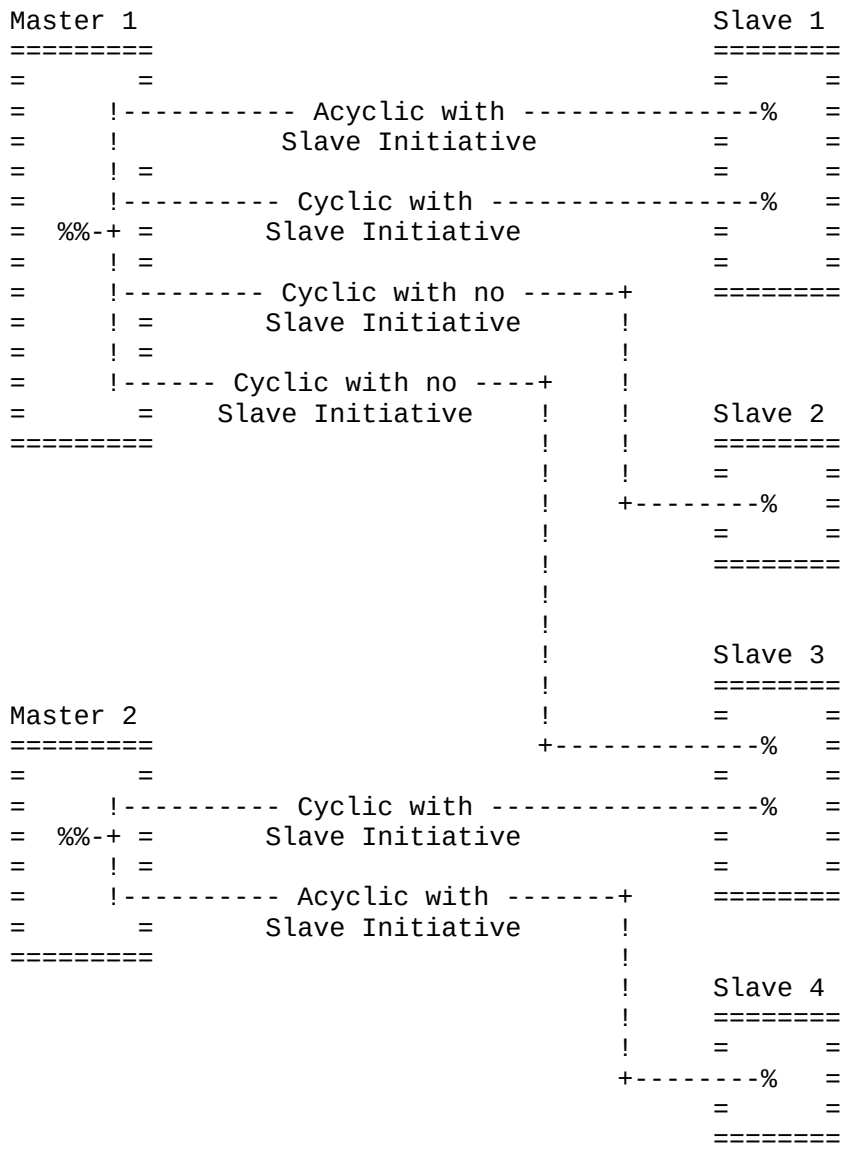
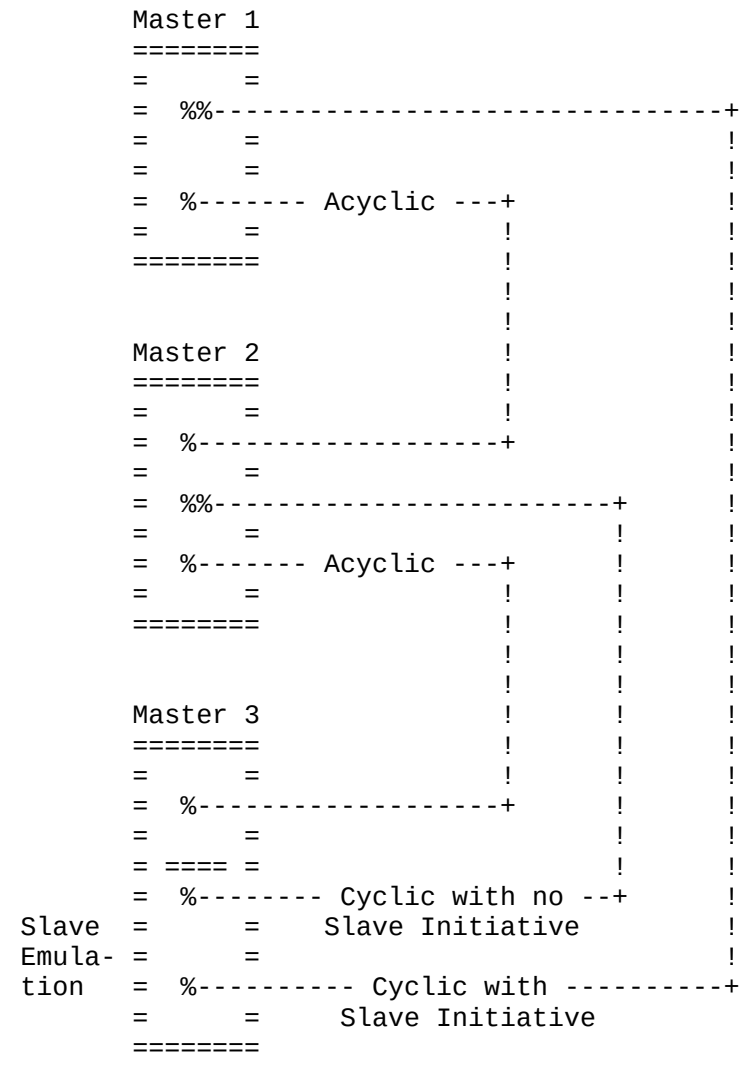


Figure 19. Communication Relationships between n Masters and n Slaves

d) n-Master <--> n-Master

An example of mutual, direct communication relationships between several masters is shown in the following figure. They occur for example in manufacturing lines, whereby the manufacturing data is passed on directly with the work piece to the next process device.



Master 3 emulates a slave on both connections for cyclic data transfer.

Figure 20. Communication Relationships between n Masters

e) Broadcast / Multicast

An example of communication relationships from one master to all other stations (broadcast), or a subset of them (multicast) is shown in the following figure. These communication relationships are used for example for the synchronization of stations or for the transmission of the time or global alarms.

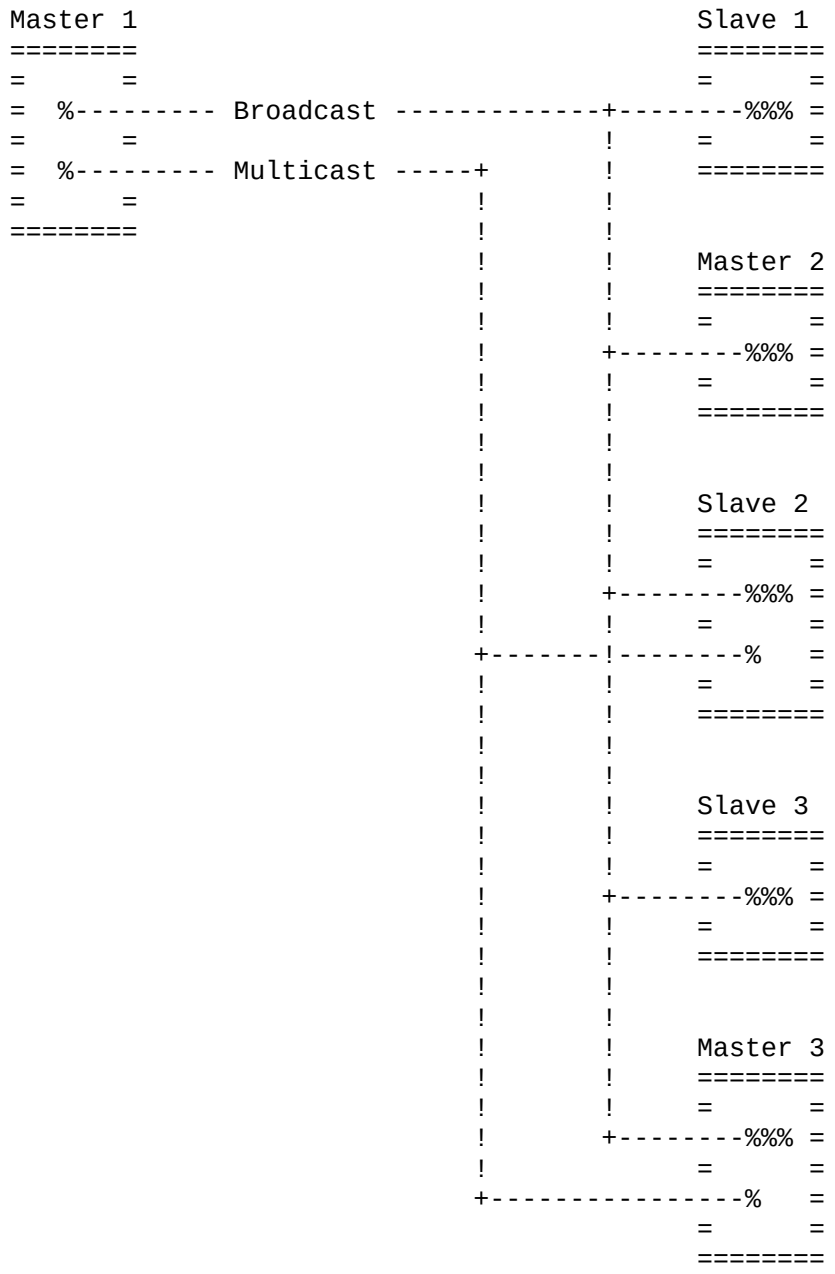


Figure 21. Broadcast / Multicast Communication Relationships

6.2.3 Interface between the LLI User and LLI

This clause describes the services, their parameters and the sequences of service primitives at the interface between the LLI user and LLI.

The LLI services are processed via LLI service access points (LLI SAPs). There are 2 LLI service access points. The following assignment shall apply:

LLI SAP 0: LLI user = FMS

LLI SAP 1: LLI user = FMA7

The detailed functionality of the services and the sequences at the interface to Layer 2 and over the bus are described from the chapter connection-oriented communication relationships on.

6.2.4 Overview of Services

The LLI provides the following services at the interface to the LLI user:

- Associate (ASS) : establishment of a connection
- Data Transfer Confirmed (DTC) : data transfer user confirmed
- Abort (ABT) : release of a connection

Optionally, the LLI may additionally provide the following services at the interface to the LLI user:

- Data Transfer Acknowledged (DTA): data transfer LLI confirmed
- Data Transfer Unconfirmed (DTU) : unconfirmed data transfer

Thereby the following definitions are valid:

The LLI user, which issues a service request to its LLI, is called responder. The LLI user, which receives a service indication for unconfirmed LLI user services from its LLI, is called receiver.

The services are described in an abbreviated, tabular form.

The following abbreviations are used in the tables:

.req	:	request service primitive
.ind	:	indication service primitive
.res	:	response service primitive
.con	:	confirmation service primitive
M	:	parameter is Mandatory for the primitive
S	:	parameter is a Selection from a set of two or more possible parameters
C	:	parameter is Conditional upon another parameter

Subparameters are indented.

Associate (ASS)

The Associate service is mandatory. It shall permit the establishment of a logical LLI connection in conjunction with the transmission of the initiate PDUs of the LLI user. A logical LLI connection is the prerequisite for the execution of the services DTC and DTA.

Table 1. Associate (ASS)

Parameter Name	.req .ind	.res .con
Argument	M	
Communication Reference	M	
LLI SAP	M	
Data	M	
Result(+)		S
Communication Reference		M
LLI SAP		M
Data		M
Result(-)		S
Communication Reference		M
LLI SAP		M
Data		M

Argument
The argument shall contain the parameters of the ASS.req and ASS.ind primitives.

Communication reference
The parameter specifies the identifier of the associated connection in the LLI CRL.

LLI SAP
The parameter specifies the LLI service access point over which the service is executed.

Data
The parameter contains the LLI user PDU (INITIATE_REQ_PDU or FMA7-INITIATE_REQ_PDU).

Result(+)
The parameter shall indicate that the Associate service was executed successfully.

Data
The parameter contains the LLI user PDU (INITIATE_RES_PDU or FMA7-INITIATE_RES_PDU).

Result(-)
The parameter shall indicate that the Associate service failed.

Data
The parameter contains the LLI user PDU (INITIATE_NRS_PDU or FMA7-INITIATE_NRS_PDU).

The following abbreviations are used in the sequences below (see the following five figures):

```
 R+    :  Result(+)
 R-    :  Result(-)
 --->  :  direction of the service primitives
 <---
 <...> :  parameter of a service primitive
 CREF  :  Communication REFerence
 LG    :  Locally Generated
 ID    :  IDentifier
 RC    :  Reason Code
 AD    :  Additional Detail
```

Execution of the Associate service at the LLI user - LLI interface (the service primitives are abbreviated as ASS.xxx):

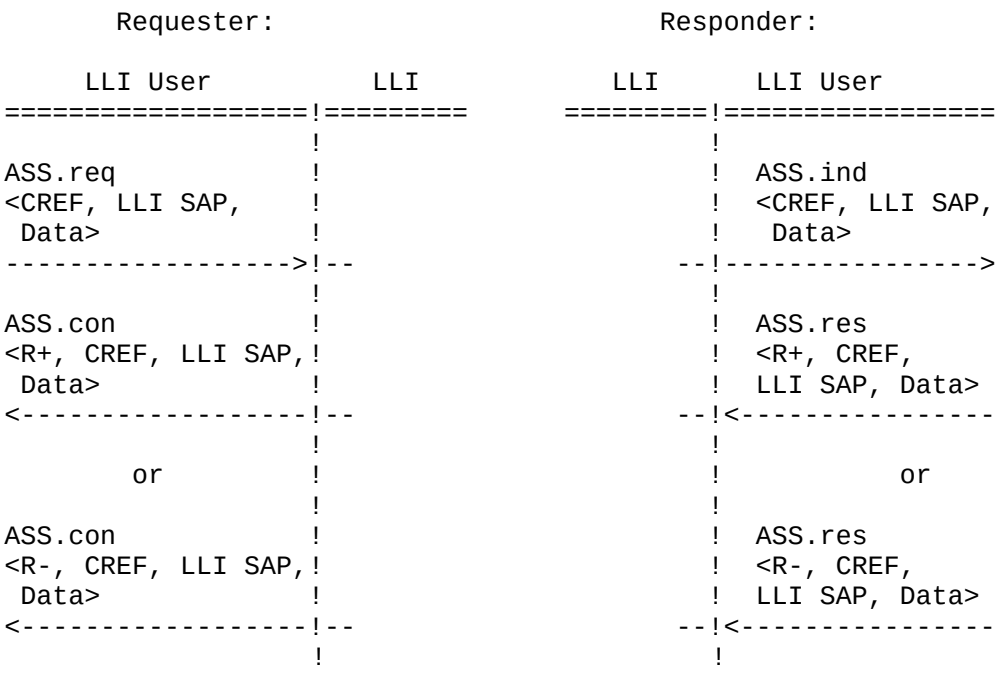


Figure 22. Associate Sequence

Data Transfer Confirmed (DTC)

The service Data Transfer Confirmed is mandatory. It permits the transmission of the LLI user PDUs for confirmed LLI user services with exception of Initiate PDUs on connection-oriented communication relationships.

Table 2. Data Transfer Confirmed

Parameter Name	.req	.res	.ind	.con
Argument	M			
Communication Reference	M			
LLI SAP	M			
Data	M			
Result			M	
Communication Reference			M	
LLI SAP			M	
Data			M	

Argument
The argument contains the parameters of the DTC.req and DTC.ind primitives.

Communication reference
The parameter shall specify the identifier of the associated connection in the LLI CRL.

LLI SAP
The parameter specifies the LLI service access point over which the service is executed.

Data
The parameter contains the LLI user PDU (confirmed Request PDU).

Result
The parameter shall indicate that the DTC service has been executed.

Data
The parameter contains the LLI user PDU (confirmed Response PDU).

Execution of the service Data Transfer Confirmed at the LLI user - LLI interface (the service primitives are abbreviated as DTC.xxx):

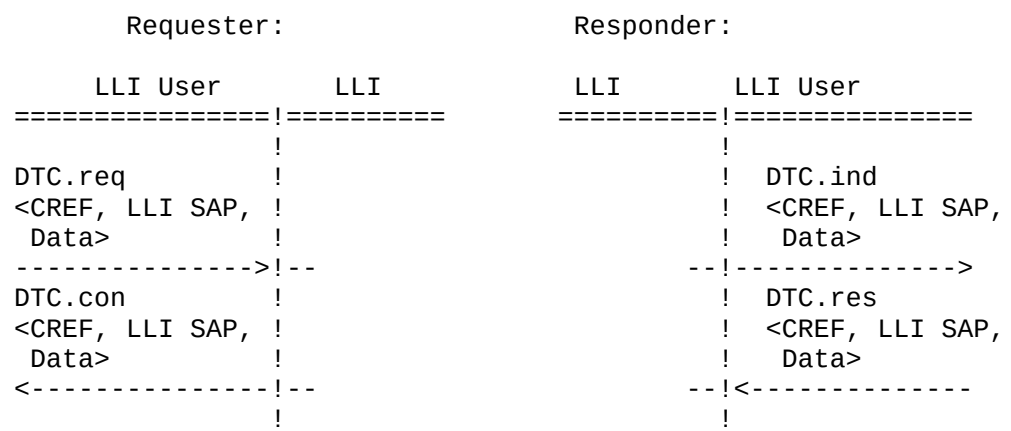


Figure 23. DTC Sequence

Data Transfer Acknowledged (DTA)

The service Data Transfer Acknowledged is optional. It permits the transmission of the LLI user PDUs for unconfirmed LLI user services over connection-oriented communication relationships.

Table 3. Data Transfer Acknowledged

Parameter Name	.req	.ind
Argument	M	
Communication Reference	M	
LLI SAP	M	
Priority	M	
Data	M	

Argument
The argument contains the parameters of the DTA.req and DTA.ind primitives.

Communication reference
The parameter specifies the identifier of the associated connection in the LLI CRL.

LLI SAP
The parameter specifies the LLI service access point over which the service is executed.

Priority
The parameter shall indicate the priority of the DTA service. It may take the value <LOW> (low priority) or <HIGH> (high priority).

Data
The parameter contains the LLI user PDU (unconfirmed request PDU).

Execution of the service Data Transfer Acknowledged at the LLI user - LLI interface (the service primitives are abbreviated as DTA.xxx).

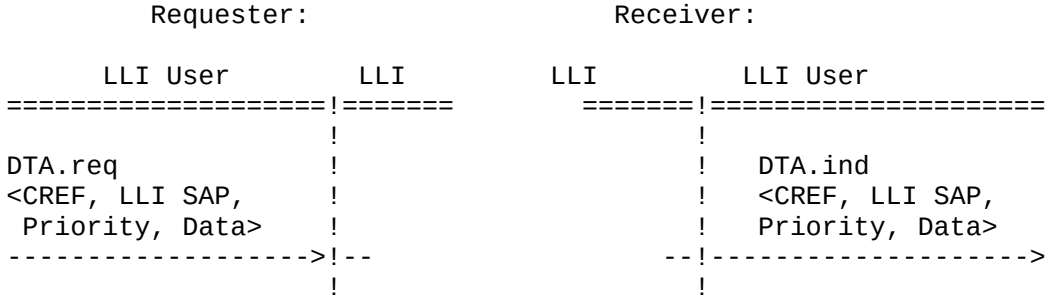


Figure 24. DTA Sequence

Data Transfer Unconfirmed (DTU)

The service Data Transfer Unconfirmed is optional. It permits the transmission of the LLI user PDUs for unconfirmed LLI user services on connectionless communication relationships.

Table 4. Data Transfer Unconfirmed

Parameter Name	.req	.ind
Argument	M	
Communication Reference	M	
LLI SAP	M	
Priority	M	
Data	M	

Argument
The argument contains the parameters of the DTU.req and DTU.ind primitives.

Communication reference
The parameter specifies the identifier of the associated connection in the LLI CRL.

LLI SAP
The parameter specifies the LLI service access point over which the service is executed.

Priority
The parameter shall indicate the priority of the DTU service. It may take the value <LOW> (low priority) or <HIGH> (high priority).

Data
The parameter contains the LLI user PDU (unconfirmed request PDU).

Execution of the service Data Transfer Unconfirmed at the LLI user - LLI interface (the service primitives are abbreviated as DTU.xxx).

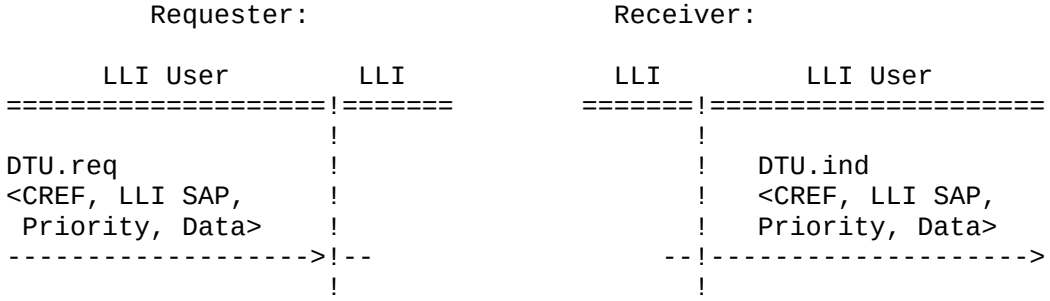


Figure 25. DTU Sequence

Abort (ABT)

The service Abort is mandatory. It permits the release of a logical LLI connection. Execution of the services DTC and DTA is only permissible after a new connection establishment.

Table 5. Abort

Parameter Name	.req	.ind
Argument	M	M
Communication Reference	M	M
LLI SAP	M	M
Locally Generated		M
Identifier	M	M
Reason Code	M	M
Additional Detail	C	C

Argument
The argument contains the parameters of the ABT.req and ABT.ind primitives.

Communication reference
The parameter specifies the identifier of the associated connection in the LLI CRL.

LLI SAP
The parameter specifies the LLI service access point over which the service is executed.

Locally Generated
The parameter shall indicate whether the Abort was initiated by the local LLI (local) or by the communication partner (remote).

Identifier
The parameter shall indicate where the reason for the connection abort was identified (user, LLI user, LLI or Layer 2).

Reason-Code
The parameter specifies the reason for the connection abort.

Additional Detail
The parameter depends on ID and RC. It contains additional information about the connection release.

Execution of the Abort service at the LLI user - LLI interface (the service primitives are abbreviated as ABT.xxx).

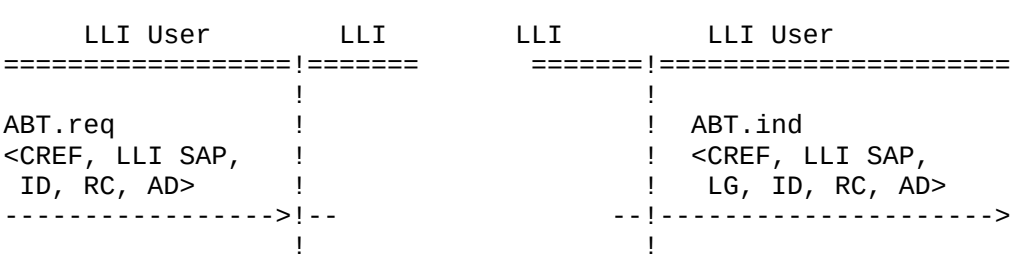


Figure 26. ABT Sequence

6.2.5 Interface between LLI and FMA7 for local functions

In this clause the services and their parameters at the interface between FMA7 and LLI are specified.

Overview of Services

The LLI provides the following services at the interface to FMA7:

- LLI-Reset : reset of the LLI
- LLI-Fault : error notification of the LLI

Optionally, the LLI may additionally provide the following services at the interface to FMA7:

- LLI-Disable : disabling of the LLI
- LLI-Load-CRL : loading of the LLI CRL
- LLI-Read-CRL : reading of the LLI CRL
- LLI-Enable : enabling of the LLI
- LLI-Ident : identification of the LLI

LLI-Reset

The service LLI Reset is mandatory. With a LLI Reset.req primitive the FMA7 causes a restart of the LLI as at power-on. This service is always permissible, independently of the current LLI state. Immediately after receiving a LLI Reset.req primitive the LLI issues a LLI Reset.con primitive to the FMA7 which indicates the beginning of the restart.

Table 6. LLI-Reset

Parameter Name	.req	.con
Argument	M	
Result		M

Argument

The argument contains no parameters.

Result

This parameter shall indicate that the LLI-Reset service has been executed.

LLI-Disable

The service LLI-Disable is optional. With the LLI-Disable service the LLI is disabled for data communication on all communication relationships with the exception of the management connection (CREF 1). All currently established connections with the exception of the management connection shall be released.

After receiving a LLI-Disable.req primitive the LLI cancels all LLI state machines related to the CREF and associated services, except for the management CREF machines (see connection establishment definition). Subsequently, the LLI deactivates all activated LSAPs (0, 2 to 63, NIL). Then the LLI state changes to LLI-DISABLE (see start of LLI definition) and a LLI-Disable.con primitive is issued to FMA7.

Table 7. LLI-Disable

Parameter Name	.req	.con
Argument	M	
Result		M

Argument

The argument contains no parameters.

Result

This parameter shall indicate that the LLIDisable service has been executed.

LLI-Load-CRL

The service LLI-Load-CRL is optional. The FMA7 delivers the LLI CRL header or the static part of a LLI CRL entry to the LLI with the service LLI Load CRL (see LLI CRL definition). At the time of delivery of the LLI-Load-CRL.req primitive the LLI shall be in the state LLI-DISABLE or in the state LOADING-CRL (see LLI state machine definition). After the execution of the service the LLI issues a LLI-Load-CRL.con primitive to FMA7.

Table 8. LLI Load CRL

Parameter Name	.req	.con
Argument	M	
LLI CRL Header	S	
LLI CRL Entry	S	
Result(+)		S
Result(-)		S
Error Type		M

Argument

The argument contains the parameters of the LLI-Load-CRL.req primitive.

LLI CRL header

This parameter contains the header of the LLI CRL.

LLI CRL entry

This parameter contains the static part of a LLI CRL entry.

Result(+)

The result(+) parameter shall indicate that the LLI-Load-CRL service was executed successfully.

Result(-)

The result(-) parameter shall indicate that the LLI-Load-CRL service failed. In this case neither the LLI header, nor a LLI CRL entry is written into the LLI CRL.

Error Type

This parameter contains information on why the LLI-Load-CRL service failed. The parameter may take the following values:

Table 9. Error Codes

Code	Meaning
LR	Service could not be executed, as resources not sufficient
SC	Service not allowed in this state (state conflict)
IV	Parameter error (static part of the LLI CRL Entry invalid).

LLI-Enable

The service LLI-Enable is optional. The loading of the LLI CRL shall be terminated with the service LLI-Enable. This service is only permitted in the state LOADING-CRL.

Upon receipt of a LLI-Enable.req primitive in the state LOADING-CRL the LLI requests the necessary resources to create the dynamic part of the LLI CRL (see LLI CRL definition). If the necessary resources are present, the LLI initializes the dynamic part of the LLI CRL. Then the LLI issues a LLI-Enable.con (R+) to FMA7. If the service failed, the LLI changes the current state to LLI-START and issues an LLI-Enable.con (R-) to FMA7. In this case the LLI remains disabled for data communication on all communication relationships with the exception of the management connection (CREF 1).

Table 10. LLI Enable

Parameter Name	.req	.con
Argument	M	
Result(+)		S
Result(-)		S
Error Type		M
Error CREF		C

Argument

The argument contains no parameters.

Result(+)

The result(+) parameter shall indicate that the LLI-Enable service was executed successfully.

Result(-)

The result(-) parameter shall indicate that the LLI-Enable service failed.

Error Type

This parameter contains information on why the LLI-Enable service failed. The parameter may take the following values:

Table 11. Error Codes

Code	Meaning
LR	Service could not be executed, as resources not sufficient
SC	Service not allowed in this state (state conflict)
IV	no valid CRL available (CRL invalid)

Error CREF

The parameter specifies at which CREF the LLI-Enable service was aborted by the LLI. The parameter is only present for the Error Type = LR.

LLI-Read-CRL

The service LLI-Read-CRL is optional. The LLI CRL header or a LLI CRL entry (subset of the static and dynamic part of the LLI CRL) shall be read with the service LLI-Read-CRL. The communication reference 0 shall be used to read the LLI CRL header. The corresponding communication reference shall be given to read a LLI CRL entry. If this communication reference is not in use, the service returns LLI-Read-CRL.con(R-).

Table 12. LLI-Read-CRL

Parameter Name	.req	.con
Argument	M	
Communication Reference	M	
Result(+)		S
LLI CRL Data		M
Result(-)		S
Error Type		M

Argument

The argument contains the parameters of the LLI-Read-CRL.req primitive.

Communication reference

The value 0 shall be given here to read the LLI CRL header. For reading the LLI CRL entry this parameter specifies the communication reference of the LLI CRL entry.

Result(+)

The result(+) parameter indicates that the LLI-Read-CRL service was executed successfully.

LLI CRL Data

This parameter contains the LLI CRL header or a subset of the LLI CRL entry.

If the parameter contains the LLI CRL header, the structure and the meaning are specified as in the LLI CRL structure definition (Structure of the LLI CRL).

If the parameter contains a subset of a LLI CRL entry with CREF > 0, the structure is as follows:

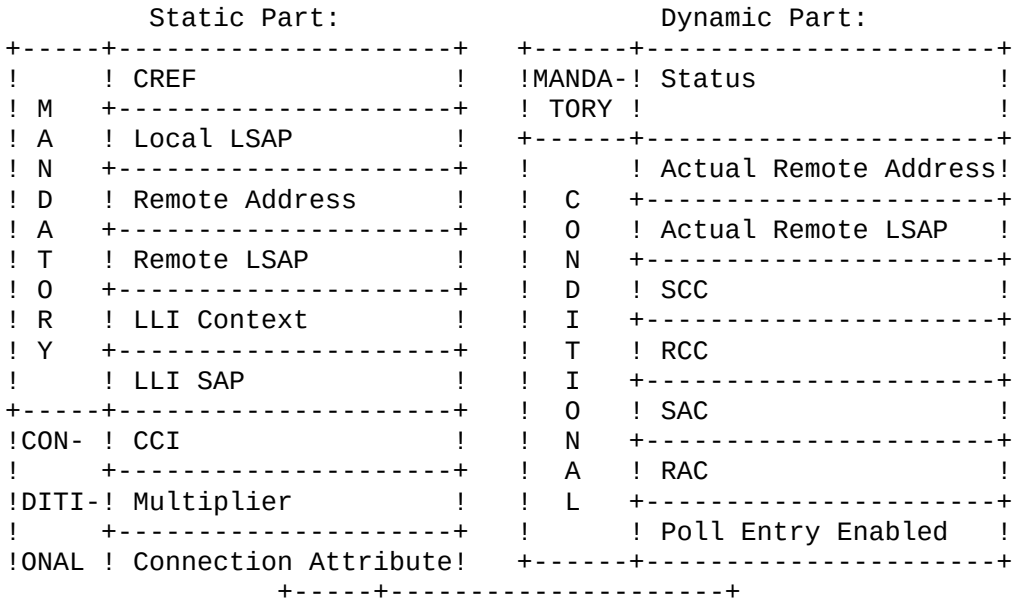


Figure 27. Structure of the excerpt from the LLI CRL Entry (Structure of the LLI CRL)

The meaning of the individual attributes corresponds to the specifications in the LLI CRL structure definition.

Result(-)

The Result (-) parameter indicates that the service LLI-Read-CRL failed.

Error Type

This parameter contains information on why the LLI-Read-CRL service failed. The parameter may take the following values:

Table 13. Error Code

Code	Meaning
NE	LLI CRL entry not available (non existent)

LLI-Ident

The service LLI-Ident is optional. It permits FMA7 to request vendor and release of the LLI software and controller type and release of the LLI hardware.

After receiving the LLI-Ident.req primitive the LLI issues the requested identification report to FMA7.

Table 14. LLI-Ident

Parameter Name	.req	.con
Argument	M	
Result		M
Vendor Name		M
Software Release		M
Controller Type		M
Hardware Release		M

Argument

The argument contains no parameters.

Result

The Result parameter shall indicate that the LLI-Ident service was executed.

Vendor Name

This parameter states the vendor name of the LLI software.

Software Release

This parameter states the revision of the LLI software.

Controller Type

This parameter states the vendor and the hardware, on which the LLI is implemented.

Hardware Release

This parameter states the revision of the hardware on which the LLI is implemented.

LLI-Fault

The Service LLI-Fault is mandatory. It is used by the LLI to indicate a fatal error to FMA7.

Table 15. LLI-Fault

Parameter Name	.ind
Argument	M
Communication Reference	M
Reason Code	M
Additional Detail	C

Argument

The argument contains the parameters of the LLI-Fault.ind primitive.

Communication reference

This parameter specifies the identifier of the related communication reference in the LLI CRL.

If an error cannot be assigned to a CREF, the value NIL shall be given here.

Reason Code

This parameter contains the error number. The values of the reason codes are specified in the formal LLI state machine definition.

Additional Detail

This parameter depends on the Reason Code and contains additional details about the reason of error.

6.2.6 LLI Communication Relationship List (LLI CRL)

The Communication Relationship List (CRL) of a station shall contain the description of all communication relationships to the other PROFIBUS stations independent of the time of usage. The CRL is created individually for every PROFIBUS station at the time of configuration. It is loaded with network management services (FMA7) or it is locally present. Every CRL entry completely describes a communication relationship. An entry is uniquely selected by the communication reference (CREF).

That part of the CRL which is relevant for the LLI is called LLI CRL. The LLI CRL contains the assignments between the communication references (CREF), the addressing of Layer 2 and that of LLI. Moreover, the communication contexts, the control mechanisms etc. are configured in the LLI CRL.

6.2.6.1 Structure of the LLI CRL

The LLI CRL consists of a header, a static and a dynamic part.

The header of the LLI CRL contains information about the number of the LLI CRL entries, the control interval for the connection establishment and release and, if required, the Poll List LSAP. The LLI CRL header is stored at CREF 0.

The static part is defined during configuration and describes the static attributes of a communication reference. The static part of the LLI CRL may be located in a ROM.

The dynamic part of the LLI CRL contains the dynamic attributes of a connection. This part is controlled only by the LLI and shall be located in a random access memory. Some attributes in the dynamic part of the LLI CRL may be omitted or remain unused depending of the type of communication reference.

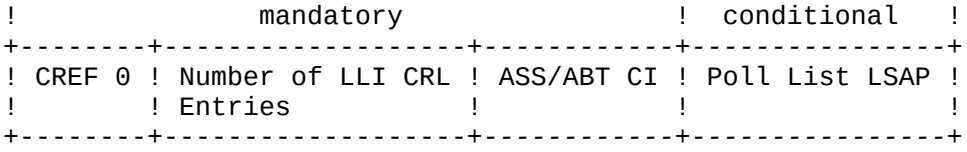


Figure 28. Structure of the LLI CRL Header

Meaning of the attributes of the LLI CRL headers:

Number of LLI CRL Entries
It specifies the number of entries in the LLI CRL. The CREFs of the entries starts with CREF 2 and shall be continued in sequence without gaps. The CREF 1 shall be additionally configured in the CRL if the station supports remote management functions as a server (the default management connection is configured in the server at CREF 1).

ASS/ABT-CI
It contains the time interval for the control of the connection establishment and connection release.

Poll List LSAP
It specifies the local Service Access Point of the Layer 2 which contains the Poll List. It is only necessary to specify this parameter at master stations which have connections for cyclic data transfer to master or slave stations, or connections for acyclic data transfer to slave stations. This attribute may take the values 0, 2 to 62 and NIL. The description and the structure of the static part of the LLI CRL (entries) is shown below.

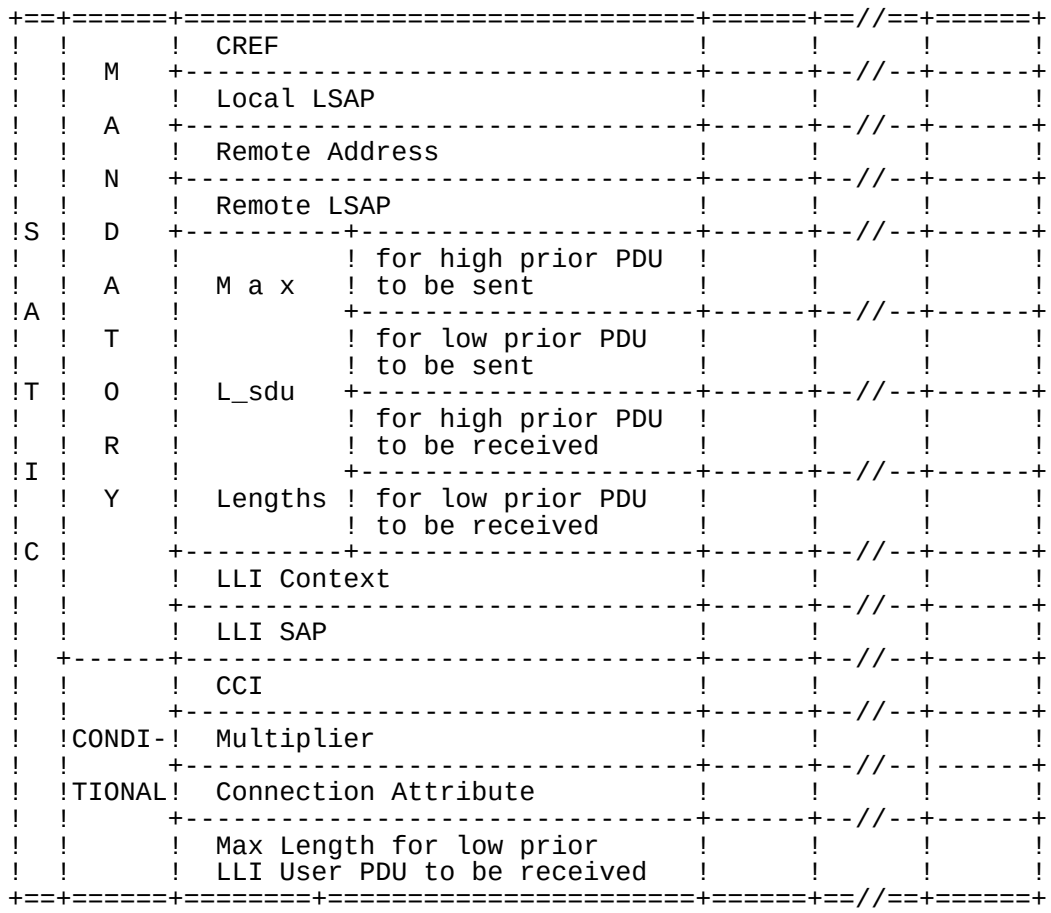


Figure 29. Structure of the LLI CRL (Entries), static Part

Meaning of the attributes in the static part of the LLI CRL:

CREF (Communication reference)
The CREF specifies a locally unique identifier for the communication reference.

Local LSAP

Specifies the local Service Access Point of Layer 2, which shall be used for this communication reference. The range of values for this attribute is shown in the following table. On connections for cyclic data transfer or master-slave connections for acyclic data transfer the Poll List LSAP shall be configured for this attribute in the LLI CRL of the master (see header of LLI CRL). For cyclic connections between two masters, that master which has not configured the Poll List SAP for this attribute shall emulate the slave.

Remote Address

It states the FDL address (Rem_add) of the remote station for this CREF. The range of values for this attribute is shown in Table 92.

Remote LSAP

It states the destination Service Access Point of the Layer 2. The range of values for this attribute is shown in table below.

Table 16. Range of Values for the Layer 2 Addressing

1)	2)	3)	4)	5)	6)	7)
range of values for the attribute Local LSAP						
0, 2 to 62, NIL	0, 2 to 62, NIL	0 to 62, NIL	0, 2 to 62, NIL	63	0, 2 to 62, NIL	0, 2 to 62, NIL
range of values for the attribute Remote Address						
0 to 126	0 to 126, All	0 to 126, All	127	0 to 126, All	127	0 to 126, All
range of values for the attribute Remote LSAP						
0 to 62, NIL	0, 2 to 62, NIL, All	0, 2 to 62, NIL, All	63	0, 2 to 62, NIL, All	0, 2 to 62, NIL	0, 2 to 62, NIL, All

Explanations:
1): requester on all connections
2): requester/responder on one Master-Master connection for acyclic data transfer
3): responder on all connection types
4): requester on one Broadcast Communication Relationship
5): receiver on one Broadcast Communication Relationship
6): requester on one Multicast Communication Relationship
7): receiver on one Multicast Communication Relationship

For connection-oriented communication relationships (see table above columns 1, 2 and 3) the specification of "requester" or "responder" relates only to the connection establishment.

Max L_sdu Lengths

The maximum Lsdu lengths (a to d) shall be calculated by FMA7 related according to user declarations and entered into the CRL. The values shall be in the range of 0 to 242 bytes.

a) Maximum L_sdu length for high priority PDUs to be sent

It indicates the maximum length of a L_sdu with which a high priority LLI PDU may be sent. The value 0 shall be specified here for communication relationships which do not use high priority LLI user services.

b) Maximum L_sdu length for low priority PDUs to be sent

It indicates the maximum length of a L_sdu with which a low priority LLI PDU may be sent.

c) Maximum L_sdu length for low priority PDUs to be received

It indicates the maximum length of a L_sdu with which a high priority LLI PDU may be received. The value 0 shall be specified here for communication relationships which do not use high priority LLI user services.

d) Maximum L_sdu length for low priority PDUs to be received

It indicates the maximum length of a L_sdu with which a low priority LLI PDU may be received.

LLI Context

The LLI context contains the attributes of a communication relationship, which shall be compatible with the corresponding attributes of the communication partner.

TYPE	max SCC	max RCC	max SAC	max RAC	ACI

Figure 30. LLI Context.

TYPE specifies the type of a communication relationship:

- MMAC : master-master connection for acyclic data transfer
- MSAC : master-slave connection for acyclic data transfer with no slave initiative
- MSAC_SI : master-slave connection for acyclic data transfer with slave initiative
- MSCY : master-slave connection for cyclic data transfer with no slave initiative (see note in subclause 4.3.1.2)
- MSCY_SI : master-slave connection for cyclic data transfer with slave initiative (see master master communication relationship)
- BRCT : Broadcast communication relationship
- MULT : Multicast communication relationship

Slave initiative (SI) means that the slave may initiate the LLI service "DTA".

max SCC (maximum value of Send Confirmed Request Counter)

On connections for acyclic data transfer at the master, this attribute specifies the permitted maximum number of parallel confirmed LLI user services to the remote communication partner (outstanding responses of the remote LLI). This corresponds to the necessary number of DTC requester state machines. In all other cases max SCC has the value 0.

max RCC (maximum value of Receive Confirmed Request Counter)

At the master (master-master communication relationship) or at the slave this attribute specifies for connections for acyclic data transfer the maximum permitted number of parallel confirmed LLI user services of the remote communica-

tion partner (outstanding responses of the LLI user). This corresponds to the necessary number of DTC responder state machines. In all other cases max RCC has the value 0.

max SAC (maximum value of Send Acknowledged Request Counter)

For all types of connections this attribute specifies the permitted maximum number of parallel unconfirmed LLI user services to the remote communication partner (outstanding acknowledges of the remote LLI). This corresponds to the necessary number of DTA requester state machines.

For slave stations with no initiative and for connectionless relationships max SAC has the value 0.

max RAC (max. value of Receive Acknowledged Request Counter)

For all types of connections this attribute specifies the maximum permitted number of unconfirmed LLI user services of the remote communication partner (outstanding free buffer). This corresponds to the necessary number of DTA acknowledge state machines).

At the master of a master-slave communication relationship with no slave initiative and for connectionless communication relationships max RAC has the value 0.

ACI (Acyclic Control Interval)

This attribute specifies the receive control interval time for the idle connection control on connections for acyclic data transfer (see subclause 4.3.4.3.2). If the connection control shall not be activated, the value 0 shall be configured here. For connections for cyclic data transfer and for all connectionless communication relationships the value 0 shall be configured for ACI.

LLI-SAP

This attribute specifies the LLI SAP which is configured for this communication relationship. If this attribute has the value 0, the assigned LLI user is FMS. If this attribute has the value 1, the assigned LLI user is FMA7. The LLI users are statically assigned to a LLI SAP (see section 2.1).

CCI (Cyclic Control Interval)

This attribute specifies the control interval time of the connection control on connections for cyclic data transfer (see subclause 4.3.4.3.2).

If this control shall not be executed at a slave, the value 0 shall be configured here.

This attribute is not necessary for connections for acyclic data transfer and for all connectionless communication relationships.

Multiplier

For connections for cyclic data transfer at the master side, this attribute specifies how often the remote address (Rem_add) and the associated remote LSAP (DSAP) of this CREF shall be entered in the Poll List. The poll interval of Layer 2 may be shortened by this means. It allows connections to be prioritized over other connections for cyclic data transfer. The range of attribute values comprises all values between 1 and 255.

The multiplier is not necessary for all other communication relationships.

Connection attribute

It contains further information about the connection type for connection-oriented communication relationships (see connection attributes). The range of values comprises the values "D", "I" and "O".

"D" : defined connection (master-master- or master-slave connection)
"I" : open connection at the requester (master-master connection)
"O" : open connection at the responder (master-master- or
master-slave connection)

This attribute is not necessary for connectionless communication relationships.

Maximum length for low priority LLI user PDUs to be received

For connections for cyclic data transfer this attribute specifies the maximum length of the low priority LLI user PDUs to be received in order to define the size of the Image Data Memory (IDM). The attribute is not necessary for all other communication relationships.

The structure of the dynamic part of the CRL (entries) is shown in the table below:

	MANDATORY		Status					
			Actual Remote Address					
			Actual Remote LSAP					
		Attributes used internally by LLI for Administration of the Communication Relationship	SCC					
			RCC					
DYNAMIC	CONAL		SAC					
			RAC					
			IMA					
			I D M	Request Invoke ID				
				LLI User PDU				
			Poll Entry Enabled					
				New				
				Old				

Figure 31. Structure of the LLI CRL (Entries), dynamic Part

Meaning of the attributes in the dynamic part of the LLI CRL:

The attributes in the dynamic part of the LLI CRL are administered only by the LLI internally. They shall not be configured.

Status

This attribute contains the status of all state machines of the communication relationship.

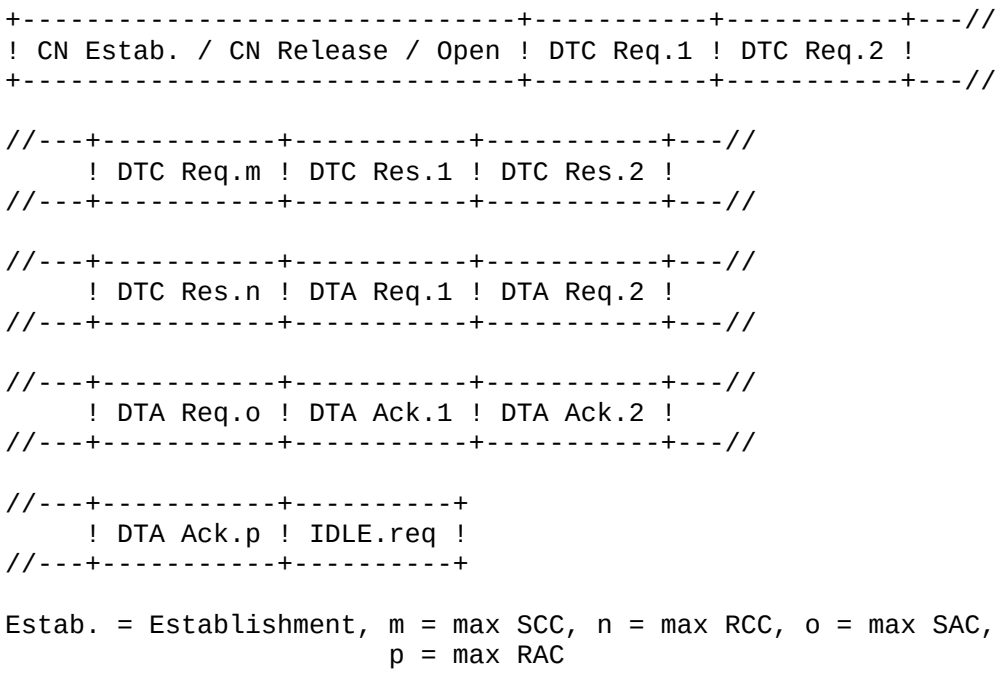


Figure 32. Structure of the Status Attribute for Connections(CN) for Acyclic Data Transfer at the Master

For master-slave connections the status attributes DTC-Res.1 to m are omitted. For master-slave connections with no slave initiative the status attributes DTA-Ack.1 to p are omitted.

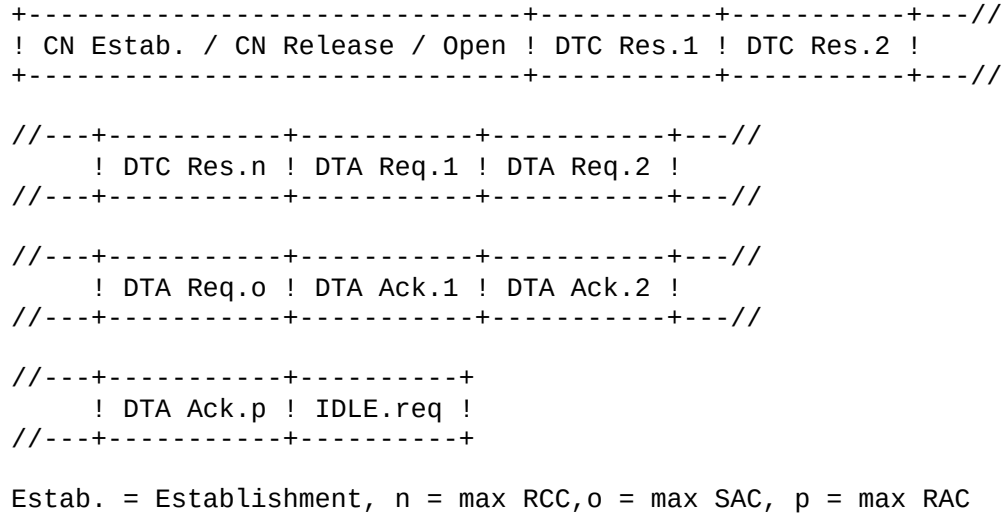


Figure 33. Structure of the Status Attribute for Connections (CN) for Acyclic Data Transfer at the Slave

For master-slave connections with no slave initiative the status attributes DTA-Req.1 to o are omitted.

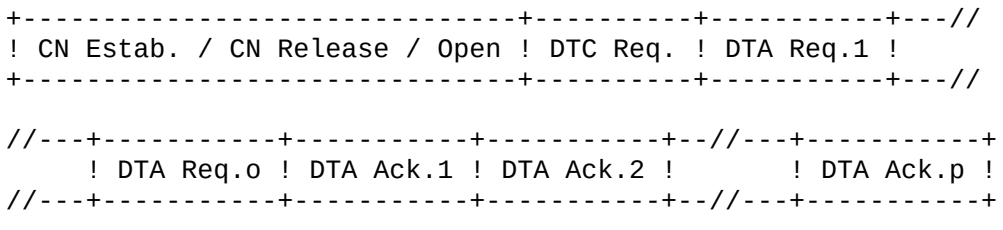


Figure 34. Structure of the Status Attribute for Connections for Cyclic Data Transfer at the Requester (Master)

For master-slave connections with no slave initiative the status attributes DTA-Ack.1 to p are omitted.

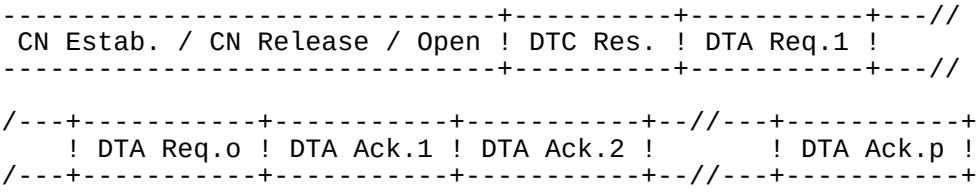
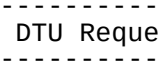


Figure 35. Structure of the Status Attribute for Connections for Cyclic Data Transfer at the Responder (Slave)

For master-slave connections with no slave initiative the status attributes DTA-Req.1 to o are omitted.

Structure of the Status Attribute for a Broadcast / Multicast Communication Relationship at the Requester:



Structure of the Status Attribute for a Broadcast / Multicast Communication Relationship at the Receiver:

Figure 36. Structure of the Status Attribute for a Connectionless Communication Relationship

Actual Remote Address

For connections with the connection attribute "O" which are not in the state "CLOSED" (see connection establishment and release), this attribute contains the FDL address (Rem_add, range of values 0 to 126) of the communication partner with has initiated the connection establishment for this CREF. In all other cases this attribute is not necessary.

Actual Remote LSAP

For connections with the connection attribute "O" which are not in the state "CLOSED", this attribute contains the local LSAP (DSAP range of values 0, 2 to 62) of the communication partner which has initiated the connection establishment for this CREF. In all other cases this attribute is not necessary.

SCC (Send Confirmed Request Counter)

For connections for acyclic data transfer at the master this attribute specifies the number of DTC requester state machines which are currently in use (i.e. not in the state "DTC-WAIT-FOR-REQ"). In all other cases SCC is not necessary.

RCC (Receive Confirmed Request Counter)

For connections for acyclic data transfer at the master (master-master communication relationships) or at the slave, this attribute specifies the number of DTC responder state machines which are currently in use (i.e. not in the state "DTC-WAIT-FOR-REQ-PDU"). In all other cases RCC is not necessary.

SAC (Send Acknowledged Request Counter)

For connections this attribute specifies the number of DTA requester state machines which are currently in use (i.e. not in the state "DTA-WAIT-FOR-REQ). For slave stations with no initiative and for connectionless communication relationships SAC is not necessary.

RAC (Receive Acknowledged Request Counter)

For connections this attribute specifies the number of DTA acknowledge state machines which are currently in use (i.e. not in the state "DTA-WAIT-FOR-REQ-PDU"). For the master of a master-slave connection with no slave initiative and for connectionless communication relationships RAC is not necessary.

IMA (Idle Machine Activated)

On connections for acyclic data transfer with connection control this attribute specifies whether the Idle state machine is in use (i.e. not in the state "IDLE-WAIT-FOR-REQ").

This attribute may take the following values:

false: Idle State machine is in the state "IDLE-WAIT-FOR-REQ".

true : Idle state machine is not in the state "IDLE-WAIT-FOR-REQ".

For all communication relationships which do not perform idle control this attribute is not necessary.

IDM (Image Data Memory)

The image data memory is only used by the LLI on connections for cyclic data transfer. For all other communication relationships this attribute is not necessary.

Request Invoke ID

The LLI of the requester stores the Invoke ID of the LLI user request PDU here.

LLI user PDU

The LLI of the requester stores the LLI user response PDU here. The LLI of the responder stores the LLI user request PDU here. The size of this attribute corresponds to the maximum LLI user PDU length for low priority messages to be received, see static part of the CRL.

Poll Entry Enabled

The attribute Poll Entry Enabled (PEE) is used by the LLI at the master of a master slave connection to administer the status of the Poll List entry of this communication reference.

This attribute may take the following values:

false: The Poll List entry in Layer 2 is locked, i.e. the registered station (Rem_add/DSAP) is not polled.

true: The Poll List entry is unlocked, i.e. the registered station (Rem_add/DSAP) is polled.

For all other communication relationships this attribute is not necessary.

New

This attribute is used by the LLI on master-slave connections for cyclic data transfer at the slave to indicate the execution of a new request (receipt of a DTC_REQ_PDU with new Invoke ID).

This attribute may take the following values:

true : A new request is being executed.
false: A current request is being executed cyclically.

Old

This attribute is used by the LLI on master-slave connections for cyclic data transfer at the slave to indicate the execution of the current request after receipt of a new request (receipt of a DTC_REQ_PDU with new Invoke ID).

This attribute may take the following values:
true : An already active order is executed after receiving a new request.
false: A current request is being executed cyclically.

For all other communication relationships this attribute is not necessary.

6.2.6.2 Connection Attribute

For connection oriented communication relationships the LLI distinguishes three types of values of the connection attribute:

"D" : Defined connection

For a defined connection the communication partner is uniquely defined by entering the remote address (Rem_add) and the assigned LSAP (DSAP) in the static part of the CRL at the time of configuration. Thereby the access protection of Layer 2 (see PROFIBUS Data Link Layer definition) is permitted in all three phases of the communication relationship.

"O" : Open connection at the Responder

For open connections no communication partner is defined at the responder at the time of configuration. In the static part of the CRL the value "ALL" is configured for the remote address and the remote LSAP. So every requester which is configured is permitted to request a connection establishment from the responder. If the LLI receives a request for connection establishment it takes the values of the parameters Loc_add/SSAP from the indication primitive of Layer 2 and enters these into the attributes Actual Remote Address and Actual Remote LSAP of the dynamic part of the LLI CRL. Then the access protection of Layer 2 is activated. Thereafter the connection behaves like a defined connection.

Upon release of the connection, it returns to the open state.

"I" : Open connection at the requester

If multiple master-master connections for acyclic data transfer use the same local LSAP at different times, they are called "open connections at the requester". At any time only one of these connections is permitted not to be in the state "CLOSED". The activation of the local LSAP takes place first in the connection establishment phase. Then this connection behaves like a defined connection. The denomination "requester" or "responder" only refers to the connection establishment.

6.2.6.3 Assignment of PDUs and Service Primitives to the Communication Reference

The LLI shall assign all PDUs which are received from Layer 2 and all service primitives which are received from the LLI - LLI user interface to a communication reference which is registered in the LLI CRL.

The LLI shall assign a PDU which is received from Layer 2 to that CREF, for which the value of the LLI SAP contained in the PDU and the values of the local LSAP, the remote address and remote LSAP contained in the FDL primitive, are equal to the values configured in the CRL. In addition for a CRL with the connection attribute “I” a FMA service primitive shall be assigned to that CRL whose state is currently not closed.

PDUs which cannot be assigned to a CREF shall be ignored by LLI.

The LLI assigns a service request, received from the LLI user, to the Layer 2 address (remote address, remote LSAP) contained in the LLI CRL entry.

The Layer 2 address is taken from the LLI CRL entry for which the values of CREF and LLI SAP of the argument are equal to the values configured in the LLI CRL.

Service primitives at the LLI - LLI user interface which cannot be assigned to a Layer 2 address shall be ignored by LLI and an error (LLI-Fault.ind) shall be issued to FMA7.

In order to allow a unique assignment the following conditions shall be fulfilled by the configuration of the CRL:

Table 17. Assignment of Remote Address, Remote LSAP and Local LSAP to the CREF

Group	Communication Relationship/ Connection Attribute	Requester: Remote Address, Remote LSAP	Requester: Local LSAP	Responder/ Receiver: Remote Address, Remote LSAP	Responder/ Receiver: Local LSAP	Req. + Res./ Receiver: Remote Address, Remote LSAP	Req. + Res./ Receiver: Local LSAP
	MMAC/D	E	E	E	E	E	E
A	MMAC/I	E	M	not allowed		not allowed	
	MMAC/O	not allowed		M (All)	E	not allowed	
	MSAC/D MSAC_SI/D MSCY/D MSCY_SI/D	E	M POLL LSAP	M	E	not allowed	
B	MSAC/I MSAC_SI/I MSCY/I MSCY_SI/I	not allowed		not allowed		not allowed	
	MSAC/O MSAC_SI/O MSCY/O MSCY_SI/O	not allowed		M (All)	E	not allowed	
C	BRCT	M	E	E	M	not allowed	
D	MULT	M	E	E	M	not allowed	

Explanations:
E : only allowed once
M : allowed several times
Req. : Requester
Res. : Responder

The groups shall not use the same values for local LSAP and remote address / remote LSAP respectively, with the exception of the remote address / remote LSAP: All.

Every communication relationship entered in the LLI CRL shall specify a unique Layer 2 address. Entries with the same Layer 2 address and different LLI SAPs are not permitted.

In the groups C and D the value All for the attributes remote address and/or remote LSAP shall only be configured if there exists for the same local LSAP no further CRL entry with a value unequal to All for this/these attributes.

6.2.6.4 Assignment of Types of Communication Relationships to the LLI User

Not all types of communication relationships are suitable for all LLI users in the same way. Therefore, for the configuration of the CRL the following rules shall apply:

Table 18. Assignment of Types of Communication Relationships to the LLI User

Type of Communication Relationship	permitted LLI User	permissible values for the Attribute LLI SAP
MMAC	FMS, FMA7	0, 1
MSAC	FMS, FMA7	0, 1
MSAC_SI	FMS	0
MSCY	FMS	0
MSCY_SI	FMS	0
BRCT	FMS	0
MULT	FMS	0

6.3 Connection-oriented Communication Relationships

For a connection-oriented communication relationship a logical one-to-one connection exists between two communication partners. An application process addresses the connection using a local CREF. The connection shall be established before it can be used for data transmission. If a connection is established and is no longer needed for data transmission then it may be released. This connection-oriented method distinguishes between the three phases:

- connection establishment
- data transfer
- connection release.

The LLI makes the different functionalities of the connection-oriented communication relationships available to the LLI users. These functionalities need not be used. LLI users may request them from the LLI as required.

6.3.1 Connection Types and Addressing

6.3.1.1 Master-Slave Communication Relationship

A connection is characterized by the mapping of the LLI user services onto Layer 2 services and their model behaviour within LLI. For a master - slave communication relationship the following kinds of connections are permitted:

- Connection for Cyclic Data Transfer with no Slave Initiative
- Connection for Cyclic Data Transfer with Slave Initiative
- Connection for Acyclic Data Transfer with no Slave Initiative
- Connection for Acyclic Data Transfer with Slave Initiative

Connection for Cyclic Data Transfer with no Slave Initiative (MSCY)

Connections for Cyclic Data Transfer with no Slave Initiative are particularly suited for application processes with the following requirements:

- Contents of variables (process data) have to be available quickly (short response times).
- Frequent access to variables.
- Relief of the application by polling within the communication and time optimized data transfer.
- Cyclic access to exactly one variable per connection.
- Slave shall be polled cyclically.

Data is transferred cyclically between a master and a slave over this connection. This kind of connection allows a particularly time optimal and efficient data transfer by storing FMS PDUs in an Image Data Memory, thus avoiding multiple transmissions of the same data. A confirmed FMS service request in the master is executed cyclically until a new confirmed FMS service request occurs or the connection is released by the master or the slave (Abort). During the data transfer phase only the confirmed FMS services Read and Write may be used. Additionally all unconfirmed FMS services for event handling with low or high priority (e.g. InformationReport) may be used in the master.

Connections for Cyclic Data Transfer with no Slave Initiative are characterized as follows:

- The master is "requester" for all allowed FMS services.
- The slave is "responder" for all allowed confirmed FMS services.
- The slave is "receiver" for all unconfirmed FMS services.
- On a connection only one confirmed FMS service is allowed at any particular time.
- Optimized data transfer, intermediate storage of PDUs
- All allowed confirmed FMS services and all unconfirmed FMS services with low priority are mapped onto the cyclic Layer 2 service CSRD with low priority.
- All unconfirmed FMS services with high priority are mapped onto the Layer 2 service SRD with high priority.
- The frequency of service execution is determined by the poll cycle within Layer 2 and the reaction time of the remote user.
- Implicit connection monitoring exists.

Connections for Cyclic Data Transfer with no Slave Initiative use the following addressing in Layer 2:

- In the master the Remote Addresses / Remote LSAPs of all connections for Cyclic Data Transfer with no Slave Initiative are entered into the Poll List of Layer 2. The Poll List is loaded into the Poll List LSAP.
- Each connection requires a LSAP in the slave.
- Several connections for Cyclic Data Transfer with no Slave Initiative may exist between a master and a slave, if several confirmed FMS services shall be executed at the same time. In this case the connections shall be assigned distinct LSAPs in the slave.

Connections for Cyclic Data Transfer with no Slave Initiative shall use LLI SAP0.

The execution of FMA7 services is not permitted on connections for Cyclic Data Transfer with no Slave Initiative.

Connection for Cyclic Data Transfer with Slave Initiative (MSCY_SI)

Connections for Cyclic Data Transfer with Slave Initiative are particularly suited for application processes with the following requirements:

- Contents of variables (process data) have to be available quickly (short response times).
- Frequent access to variables.

- Relief of the application by polling within the communication and time optimized data transfer.
- Cyclic access to exactly one variable per connection.
- Slave shall be polled cyclically.
- Slave shall indicate events.

Data is transferred cyclically between a master and a slave over this connection. This kind of connection allows a particularly time optimal and efficient data transfer by storing FMS PDUs in an Image Data Memory, thus avoiding multiple transmissions of the same data. A confirmed FMS service request in the master is executed cyclically until a new confirmed FMS service request occurs or the connection is released by the master or the slave (Abort). During the data transfer phase only the confirmed FMS services Read and Write may be used. Additionally all unconfirmed FMS services for event handling with low or high priority (e.g. InformationReport) may be used in the master and the slave.

Connections for Cyclic Data Transfer with Slave Initiative are characterized as follows:

- The master is "requester" for all allowed confirmed FMS services.
- The slave is "responder" for all allowed confirmed FMS services.
- Master and slave are either "requester" or "receiver" or both for all unconfirmed FMS services.
- On a connection only one confirmed FMS service is allowed at any time.
- Optimized data transfer, intermediate storage of PDUs
- All allowed confirmed FMS services and all unconfirmed FMS services with low priority are mapped onto the cyclic Layer 2 service CSRD with low priority.
- All unconfirmed FMS services with high priority are mapped onto the Layer 2 service SRD with high priority.
- The frequency of service execution is determined by the poll cycle within Layer 2 and the reaction time of the remote user.
- Implicit connection monitoring exists.

Connections for Cyclic Data Transfer with Slave Initiative use the following addressing in Layer 2:

- In the master the Remote Addresses / Remote LSAPs of all connections for Cyclic Data Transfer with Slave Initiative are entered into the Poll List of Layer 2. The Poll List is loaded into the Poll List LSAP.
- Each connection requires a LSAP in the slave.
- Several connections for Cyclic Data Transfer with Slave Initiative may exist between a master and a slave, if several confirmed FMS services shall be executed at the same time. In this case the connections shall be assigned distinct LSAPs in the slave.

Connections for Cyclic Data Transfer with Slave Initiative shall use LLI SAP 0.

The execution of FMA7 services is not allowed on connections for Cyclic Data Transfer with Slave Initiative.

Connection for Acyclic Data Transfer with no Slave Initiative (MSAC)

Connections for Acyclic Data Transfer with no Slave Initiative are particularly suited for application processes with the following characteristics:

- Parallel LLI user services over a connection.
- Rare access to process data (process start, diagnostic device, programming device etc.)
- Reaction time uncritical.
- Polling done by the application.
- Sporadic service request from the LLI user of the master.
- Frequently changing LLI user services over a connection.
- Transmission of diagnostic and configuration data.

Data is transferred acyclically between a master and a slave over this connection. Each service request of the LLI user leads to a single independent data exchange with a slave. All confirmed and unconfirmed LLI user services are allowed in the master. In the slave the Abort services are allowed.

Connections for Acyclic Data Transfer with no Slave Initiative are characterized as follows:

- The master is "requester" for all LLI user services.
- The slave is "responder" for all confirmed LLI user services.
- The slave is "receiver" for all unconfirmed LLI user services.
- Several confirmed and / or unconfirmed LLI user services are allowed on a connection at the same time (parallel requests).
- All confirmed LLI user services and all unconfirmed LLI user services with low priority are mapped onto the cyclic Layer 2 service CSRD with low priority.
- All unconfirmed LLI user services with high priority are mapped onto the Layer 2 service SRD with high priority.
- Explicit connection monitoring is possible.

Connections for Acyclic Data Transfer with no Slave Initiative use the following addressing in Layer 2:

- In the master the Remote Addresses / Remote LSAPs of all connections for Acyclic Data Transfer with no Slave Initiative are entered into the Poll List of Layer 2. The Poll List is loaded into the Poll List LSAP.
- Each connection requires a LSAP in the slave.
- Several connections for Acyclic Data Transfer with no Slave Initiative may exist between a master and a slave at the same time. In this case the connections shall be assigned distinct LSAPs in the slave.

Connection for Acyclic Data Transfer with Slave Initiative (MSAC_SI)

Connections for Acyclic Data Transfer with Slave Initiative are particularly suited for application processes with the following characteristics:

- Parallel LLI user services over a connection.
- Rare access to process data (process start, diagnostic device, programming device etc.)
- Reaction time uncritical.
- Polling done by the application.
- Sporadic service request from the LLI user of the master.
- Frequently changing LLI user services over a connection.
- Slave shall indicate events.

Data is transferred acyclically between a master and a slave over this connection. Each service request leads to a single independent data exchange with a slave. All confirmed LLI user services are allowed in the master. All unconfirmed LLI user services are allowed in the master and the slave.

Connections for Acyclic Data Transfer with Slave Initiative are characterized as follows:

- The master is "requester" for all confirmed LLI user services.
- The slave is "responder" for all confirmed LLI user services.
- Master and slave are either "requester" or "receiver" or both for all unconfirmed LLI user services.
- Several confirmed and / or unconfirmed LLI user services are allowed over a connection at the same time (parallel requests).
- All confirmed LLI user services and all unconfirmed LLI user services with low priority are mapped onto the cyclic Layer 2 service CSRD with low priority.
- All unconfirmed LLI user services with high priority are mapped onto the Layer 2 service SRD with high priority.
- Explicit connection monitoring is possible.

Connections for Acyclic Data Transfer with Slave Initiative use the following addressing in Layer 2:

- In the master the Remote Addresses / Remote LSAPs of all connections for Acyclic Data Transfer with Slave Initiative are entered into the Poll List of Layer 2. The Poll List is loaded into the Poll List LSAP.
- Each connection requires a LSAP in the slave.
- Several connections for Acyclic Data Transfer with Slave Initiative may exist between a master and a slave at the same time. In this case the connections shall be assigned distinct LSAPs in the slave.

Connections for Acyclic Data Transfer with Slave Initiative shall use LLI SAP 0.

The execution of FMA7 services is not permitted on connections for Acyclic Data Transfer with Slave Initiative.

6.3.1.2 Master-Master Communication Relationship

For a master-master communication relationship the following kind of connection is possible:

- Connection for Acyclic Data Transfer.

If a connection for Cyclic Data Transfer is to be established between two masters, one of these two masters (Responder / Receiver) shall emulate a slave for this communication relationship. In this case this communication relationship is looked upon as a master-slave communication relationship. It is declared as a connection for Cyclic Data Transfer with or with no Slave Initiative and is described below. Slave emulation is marked specific to the connection in the CRL by not entering the Poll List LSAP into the attribute "Local LSAP" for this connection (see also Header of CRL definition before).

Connection for Acyclic Data Transfer (MMAC)

Data is exchanged between masters over this connection. Each service request of the LLI user leads to a single independent data exchange. All confirmed and unconfirmed LLI user services are allowed.

Connections for Acyclic Data Transfer are particularly suited for application processes with the following characteristics:

- Application processes in complex PROFIBUS stations which require mutual, parallel and priority controlled data transfer.
- Rare access to process data (process start, diagnostic device, programming device etc.)
- Reaction time uncritical.
- Polling done by the application.
- Sporadic service request from the LLI user of the master.
- Frequently changing LLI user services over a connection.
- Transmission of diagnostic and configuration data.

Connections for Acyclic Data Transfer are characterized as follows:

- Each master may be "requester" and / or "responder" and / or "receiver" for all LLI user services.
- Several confirmed and / or unconfirmed LLI user services are allowed on a connection at the same time - not only parallel but also mutual.
- All LLI user services are mapped onto the Layer 2 service SDA.
- Explicit connection monitoring is possible.

Connections for Acyclic Data Transfer use the following addressing in Layer 2:

- Each master-master connection requires a LSAP.
- Several connections for Acyclic Data Transfer may exist between two masters at the same time. In this case the connections shall be assigned distinct LSAPs.

6.3.2 Connection Establishment

Connection establishment is used to install a logical connection between two communication partners for connection-oriented communication relationships. If no connection establishment has been completed for a connection-oriented communication relationship, the communication partners are not in the data transfer phase and no data exchange between the users is permissible. In this situation the LLI accepts only a request to establish the connection from its own LLI user or from the communication partner (ASS.req resp. ASS_REQ_PDU).

The FMS user may use the Initiate service of FMS to establish connections. FMS maps this service onto the Associate service (ASS) of LLI.

The FMA7 user may use the FMA7 Initiate service of FMA7 to establish connections. FMA7 maps this service onto the Associate service (ASS) of LLI.

Connection establishment within LLI is executed in the same way for the two LLI users FMS and FMA7.

The sequence charts and specifications below use the following symbols and abbreviations:

Legend for the following sequence charts and specifications:

```
IVIDn       :  Invoke ID with number n
INFO        :  InformationReport
R+          :  positive response, Result(+)
R-          :  negative response, Result(-)
CN          :  Connection
# n         :  communication reference
[...]       :  PDU (Protocol Data Unit)
]--->       :  A service primitive is called by LLI
[<---       :  Service primitive is not forwarded
(---        :  immediate confirmation
(---
--->        :  Direction of service primitives and PDU resp.
<---
\           :  Event or condition
"           :  Action
*           :  Level of a PDU
(...)       :  Service primitive (FDL) For the FDL service primitives the values
               of parameters are not shown, with the exception of <High> and
                    <Low>.)
<...>       :  Parameters or values of  parameters of a service primitive. The
               parameters at the LLI user - LLI interface are shown completely.
               At all other interfaces only those parameters are shown that are
               necessary for the understanding of the working of LLI.
INI.req     :  Initiate.req / FMA7-Initiate.req
INI.ind     :  Initiate.ind / FMA7-Initiate.ind
INI.res     :  Initiate.res / FMA7-Initiate.res
INI.con     :  Initiate.con / FMA7-Initiate.con
INI_REQ_PDU :  INITIATE_REQ_PDU / FMA7-INITIATE_REQ_PDU
INI_RES_PDU :  INITIATE_RES_PDU / FMA7-INITIATE_RES_PDU
INI_ERR_PDU :  INITIATE_ERROR_PDU / FMA7-INITIATE_ERROR_PDU
ABO.req     :  Abort.req / FMA7-Abort.req
ABO.ind     :  Abort.ind / FMA7-Abort.ind
```

The LLI user maps a request from the user to establish a connection (Initiate.req from FMS user, FMA7-Initiate.req from FMA7 user) onto the LLI service primitive ASS.req (see overview of services before). The LLI of the requester starts the monitoring of connection establishment (see connection establishment definition below) and generates an Associate Request PDU (ASS_REQ_PDU). This PDU contains the LLI context and the PDU of the LLI user. If the connection to be established is an open connection in the requester (Connection Attribute = "I"), then the assigned LSAP shall be activated with access protection (parameter Ac-

cess = Remote Address) before the ASS_REQ_PDU is passed to Layer 2 (see also the formal description of the formal state machine below).

The LLI of the responder compares the local with the remote LLI context upon receipt of the ASS_REQ_PDU. If the LLI contexts of both communication partners are compatible, an ASS.ind is passed to the LLI user which contains the INI_REQ_PDU of the remote LLI user. Additionally, the monitoring of connection establishment is started. For the establishment of an open connection in the responder (Connection Attribute = "O") the LLI of the responder shall additionally activate the assigned LSAP with access protection (parameter Access = Actual Remote Address). The LLI service primitive ASS.ind is mapped onto an Initiate.ind by FMS or onto a FMA7-Initiate.ind by FMA7 and is passed to the user (see also: description of the formal state machine below).

The LLI user maps the positive response of the user (INI.res<R+>) onto the LLI service primitive ASS.res<R+>. The LLI of the responder generates an ASS_RES_PDU, which contains the INI_RES_PDU and passes it to Layer 2 for transmission to the remote partner. After the transmission the connection is established (status "OPEN") and is in the data transfer phase. The monitoring of connection establishment is stopped and the monitoring of data transfer is started, if configured.

If the user of the responder rejects the request to establish a connection (INI.res<R->), the LLI user maps this onto the LLI service primitive ASS.res<R->. If the LLI user detects an error such as incompatible context, it also passes an ASS.res<R-> to the LLI. LLI generates an ASS_NRS_PDU, which contains the INI_ERRPDU, and passes it to Layer 2 for transmission to the remote partner. The connection is not established. For an open connection in the responder (Connection Attribute = "O") LLI shall activate the assigned LSAP with parameter Access = All after transmission of the ASS_NRS_PDU.

If the LLI of the responder detects an error during connection establishment, the connection shall not be established, but a connection release shall be performed. LLI starts the monitoring of connection release (T2, connection release definition below), generates an ABT_REQ_PDU and enters the reason for the connection release into the field Reason Code (RC, see LLI PDU definition). If the LLI contexts are incompatible, the field Additional Detail (AD) contains the LLI context of the responder. For an open connection in the responder (Connection Attribute = "O") LLI shall activate the assigned LSAP with parameter Access = All after transmission of the ABT_REQ_PDU.

After receipt of an ASS_RES_PDU from the responder the LLI of the requester stops monitoring the connection establishment and passes an ASS.con<R+>, which contains the INI_RES_PDU, to the LLI user. The connection is now established (status "OPEN") and is in the data transfer phase. The monitoring of data transfer is started, if configured (see data transfer definition). FMS maps the ASS.res<R+> onto an Initiate.con(+) and passes it to the user. FMA7 maps the ASS.res<R+> onto a FMA7-Initiate.con(+) and passes it to the user.

If the requester receives an ASS_NRS_PDU, then the responder has rejected the request to establish a connection. LLI stops monitoring the connection establishment and passes an ASS.con<R->, which contains the INI_ERR_PDU, to the LLI user. The connection is not established, but a local connection release, which is monitored, is performed. For an open connection in the requester (Connection Attribute = "I") LLI shall deactivate the assigned LSAP. FMS maps the ASS.con<R-> onto an Initiate.con(-) and passes it to the user. FMA7 maps the ASS.con<R-> onto a FMA7-Initiate.con(-) and passes it to the user.

If the requester receives an ABT_REQ_PDU, the connection establishment is stopped and a local connection release, which is monitored, is started. The reason for rejecting the request to establish a connection shall be found in the field RC.

If the reason for this rejection was an incompatibility of the LLI contexts, then the field AD contains the remote LLI context. LLI passes an ABT.ind to the LLI user (see definition of interface between LL user and LLI). For an open con-

nection in the requester (Connection Attribute = "I") LLI shall deactivate the assigned LSAP. FMS maps the ABT.ind of the LLI onto an Abort.ind and passes it to the user. FMA7 maps the ABT.ind onto a FMA7-Abort.ind and passes it to the user.

If a connection is in the data transfer phase and an ASS_REQ_PDU is received on this connection or an ASS.req is performed by the LLI user, then LLI performs a connection release (see connection release definition).

6.3.2.1 Monitoring Connection Establishment

For time monitoring of connection establishment the timer T1 is used. It is connection specific and mandatory for master and slave devices. The interval for monitoring connection establishment is configured in the header of the LLI CRL.

In the requester the timer T1 controls the arrival of the response from the remote partner (ASS_RES_PDU, ASS_NRS_PDU, ABT_REQ_PDU) after having sent the ASS_REQ_PDU. The timer T1 is started before the ASS_REQ_PDU is sent and is stopped upon receipt of the response of the remote partner.

In the responder the timer T1 controls the response of the local LLI user to the request to establish a connection from the remote partner as well as the sending of the response to the requester. The timer T1 is started after the ASS_REQ_PDU has been received and is stopped after having sent the response to the remote partner.

If the timer T1 expires in the requester or the responder, a connection release is performed (see connection release definition).

6.3.2.2 Associate for Master-Slave Communication Relationships

For master-slave communication relationships the master shall always take the initiative for connection establishment.

The LLI maps the Associate service onto the cyclic Layer 2 service CSRD. The LLI in the master shall start the Layer 2 polling to the related slave (Rem_add/DSAP). Thereby the Layer 2 service primitive FDL_CYC_POLL_ENTRY.req with parameter Marker : "unlock" is used.

For a master-slave connection for acyclic data transfer with no Slave Initiative the LLI in the master shall stop the Layer 2 polling to the related slave after connection establishment. Thereby the Layer 2 service primitive FDL_CYC_POLL_ENTRY.req with parameter Marker : "lock" is used. For all other types of master-slave connections the Layer 2 polling to the related slave remains unlocked after successful connection establishment.

EXAMPLE: The following sequence charts show examples of possible connection establishment sequences.

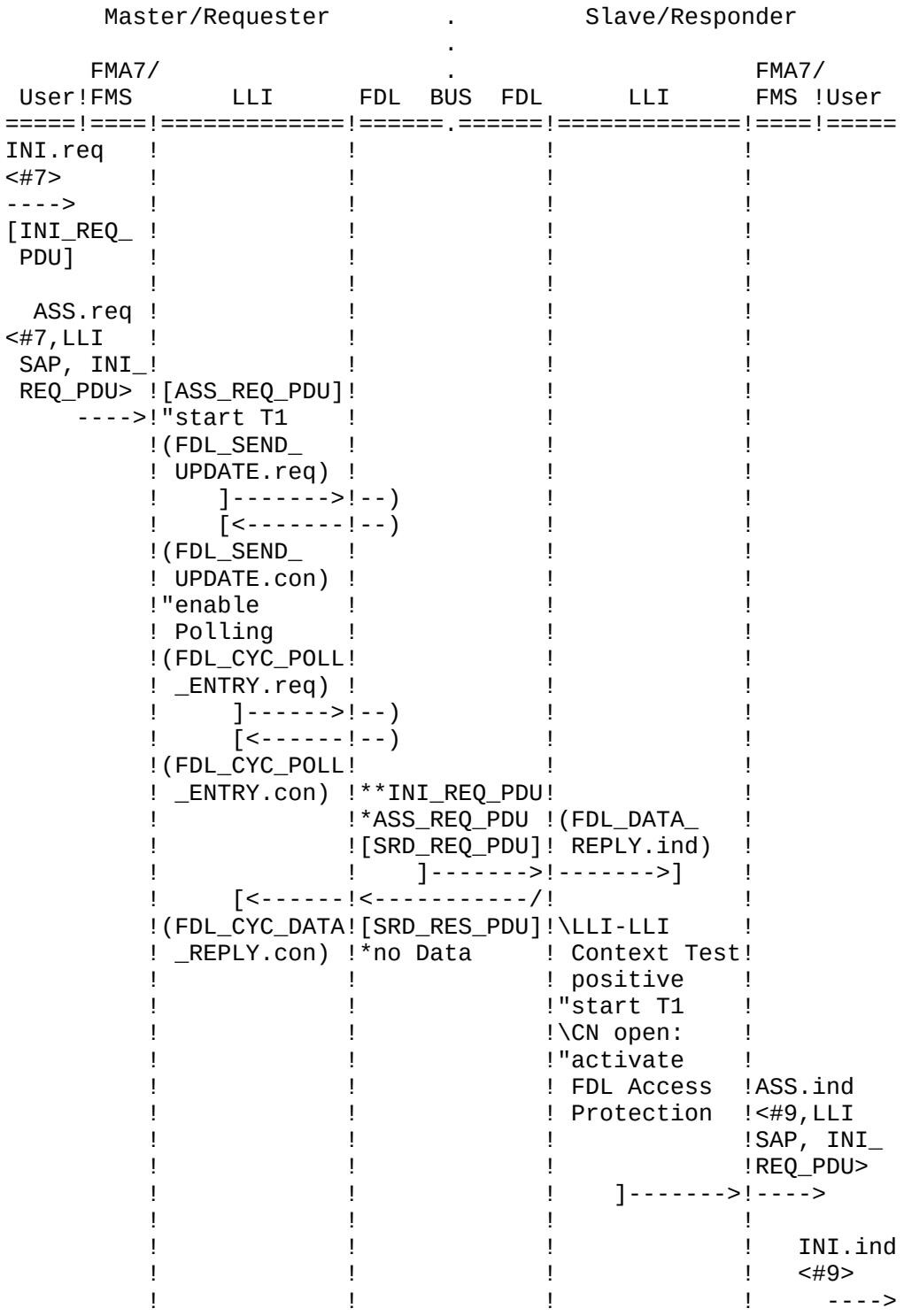


Figure 37. Master-Slave Communication Relationship / all Connection Types / Connection Establishment / Request / LLI-LLI Context Test positive

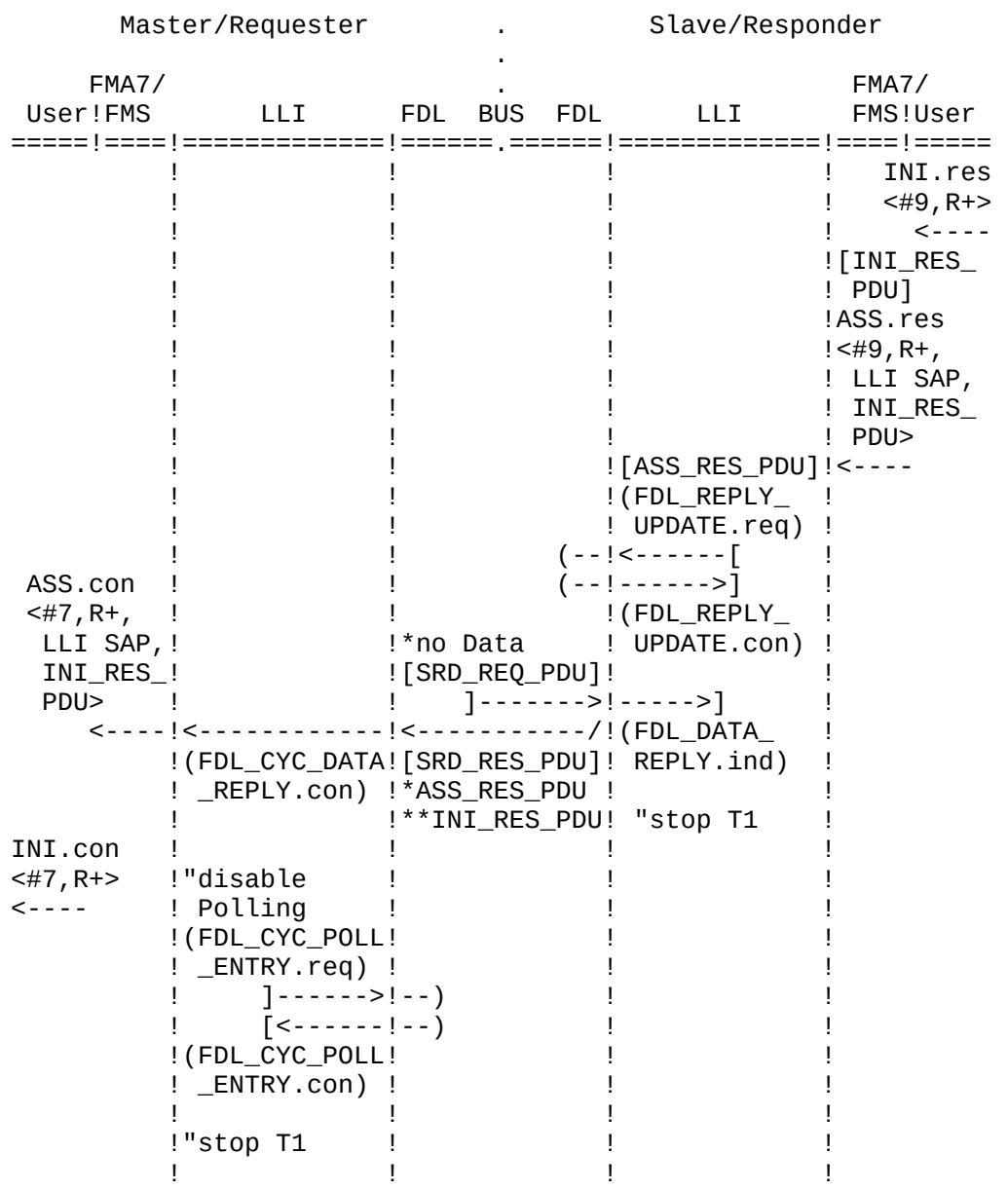


Figure 38. Master-Slave Communication Relationship / Connection for Acyclic Data Transfer with no Slave Initiative / Connection Establishment / Positive Response

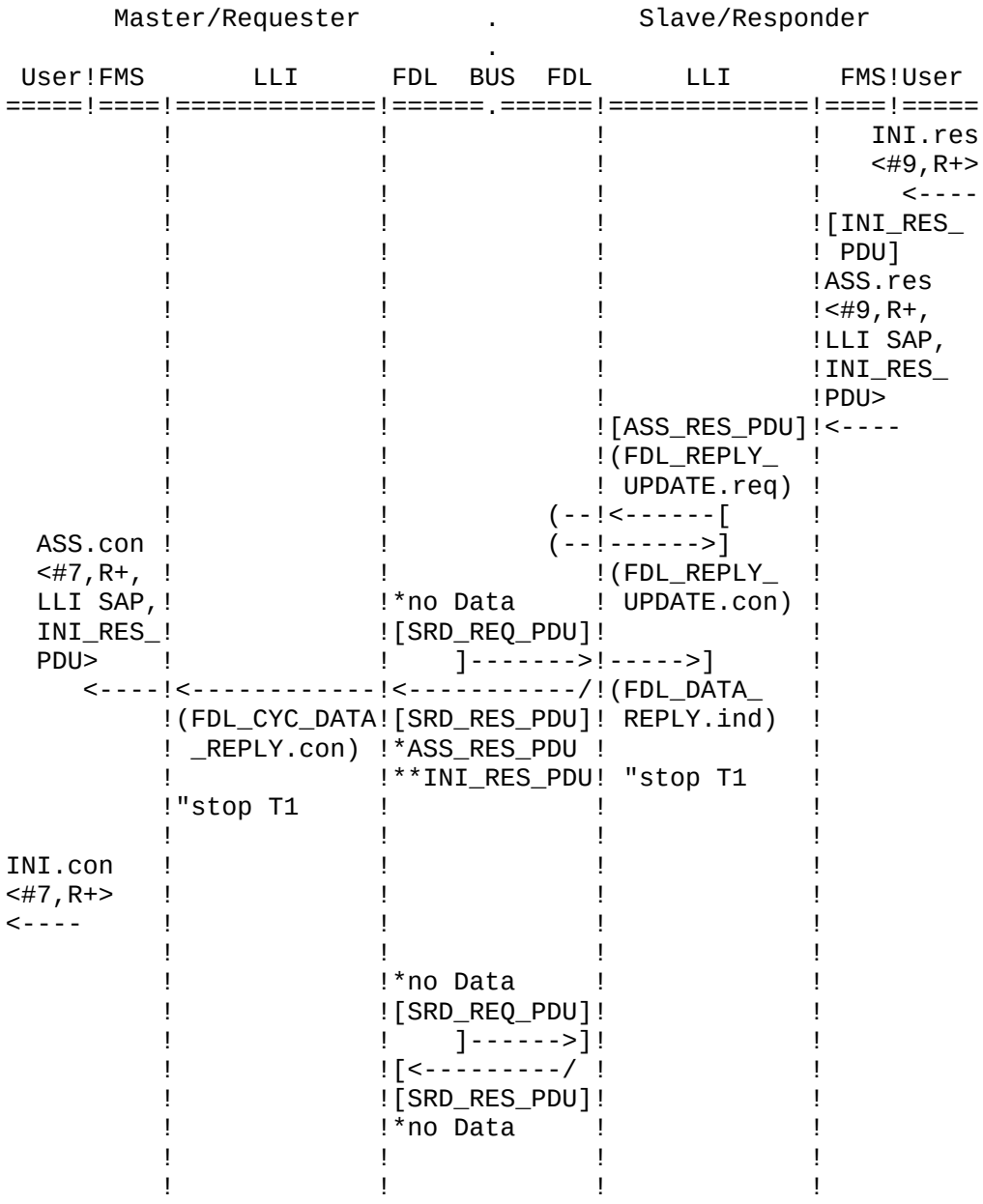


Figure 39. Master-Slave Communication Relationship / Connection for Acyclic Data Transfer with Slave Initiative or Connection for Cyclic Data Transfer / Connection Establishment / Positive Response

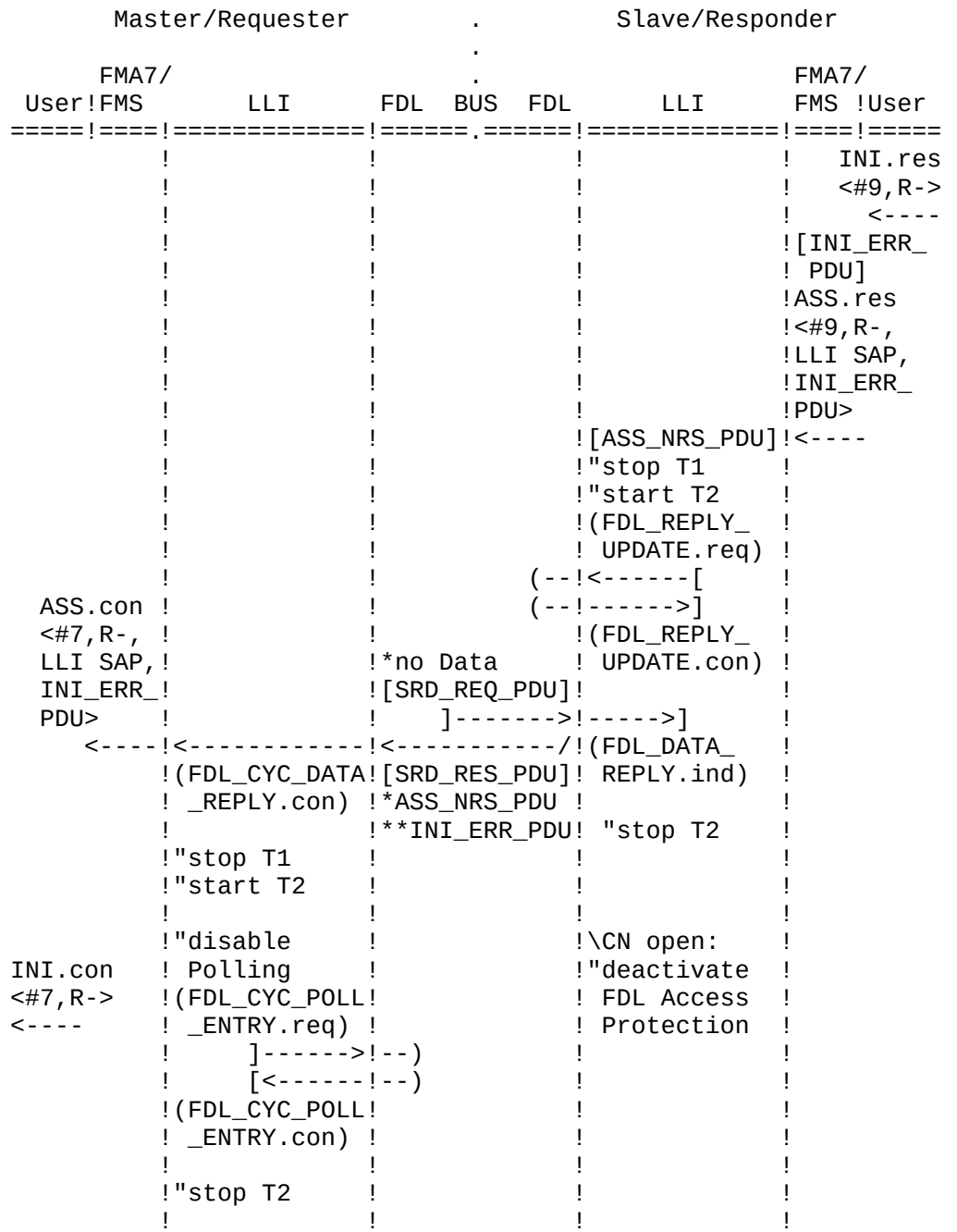


Figure 40. Master-Slave Communication Relationship / all Connection Types / Connection Establishment / Negative Response

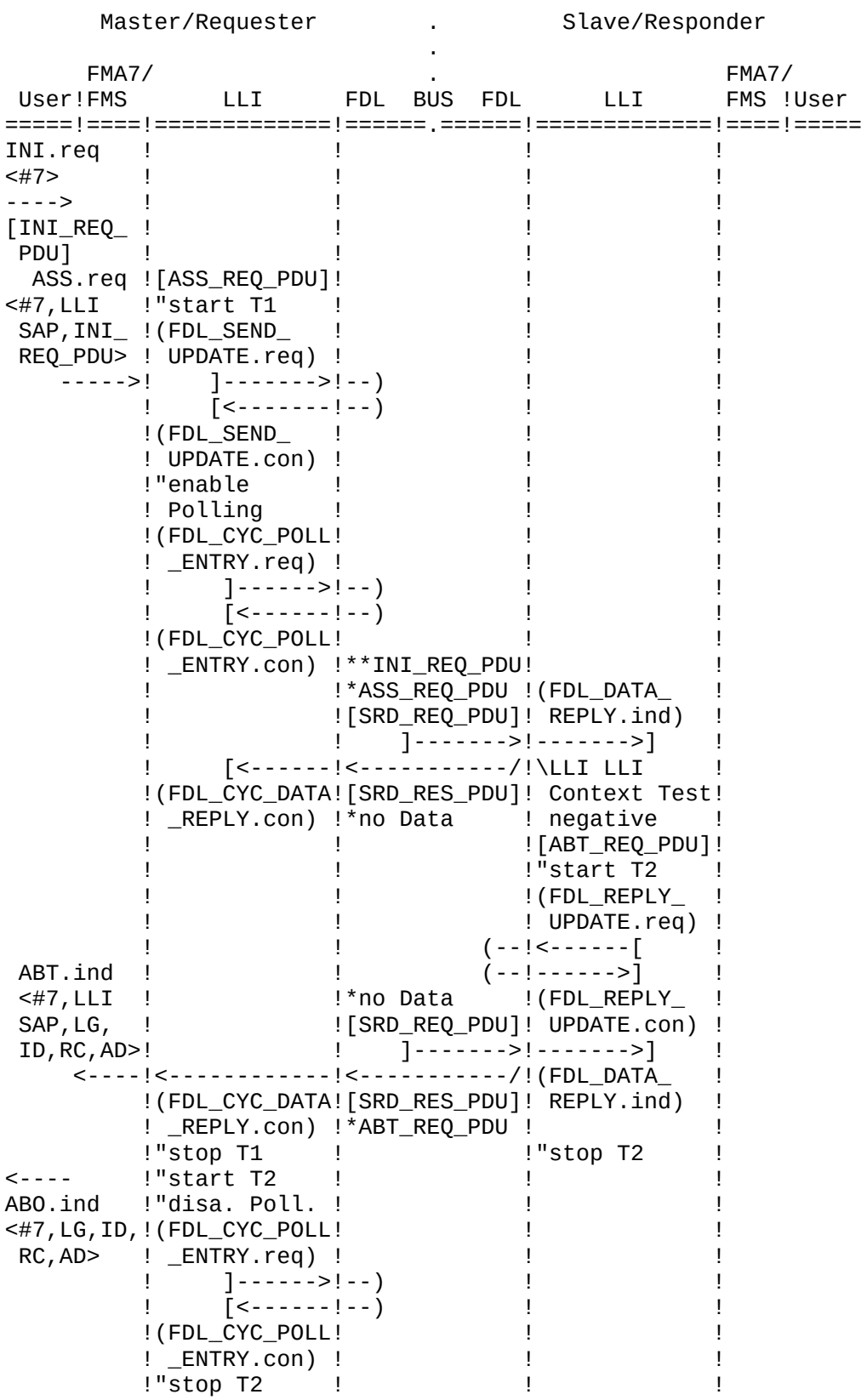


Figure 41. Master-Slave Communication Relationship / all Connection Types / Connection Establishment / LLI-LLI Context Test negative

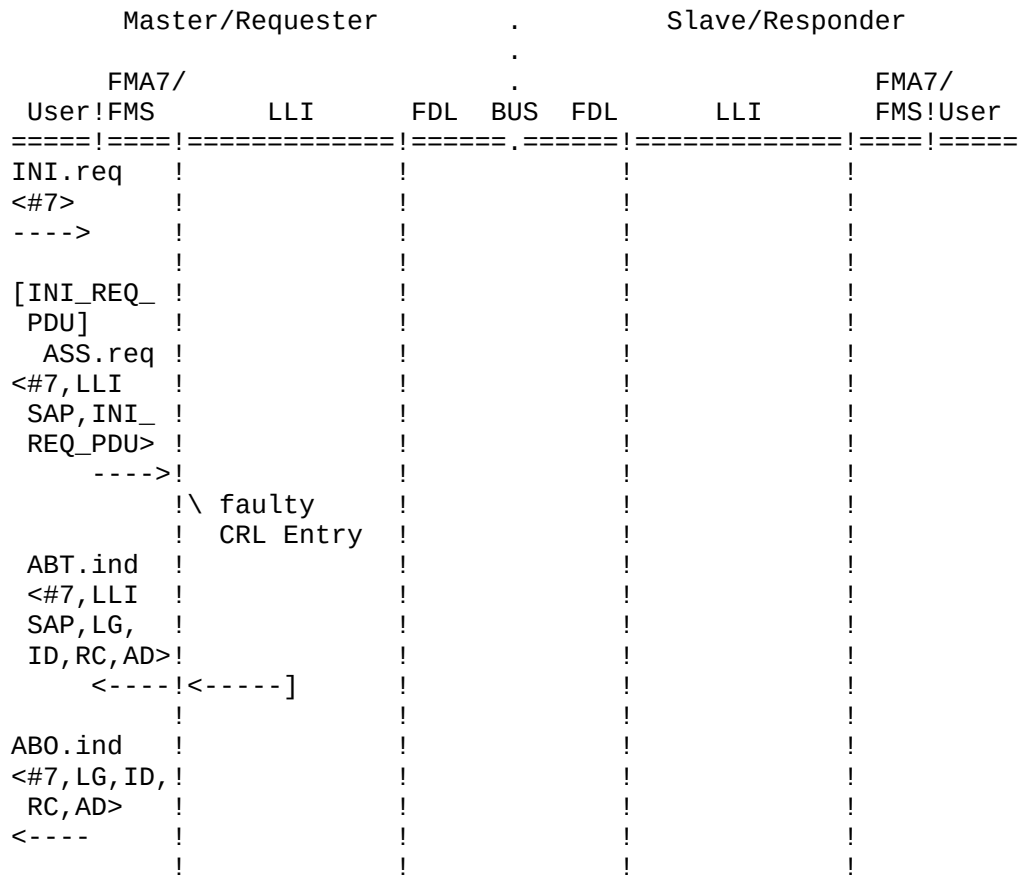


Figure 42. Master-Slave Communication Relationship / all Connection Types / Connection Establishment / Request / local Error: CRL Entry faulty

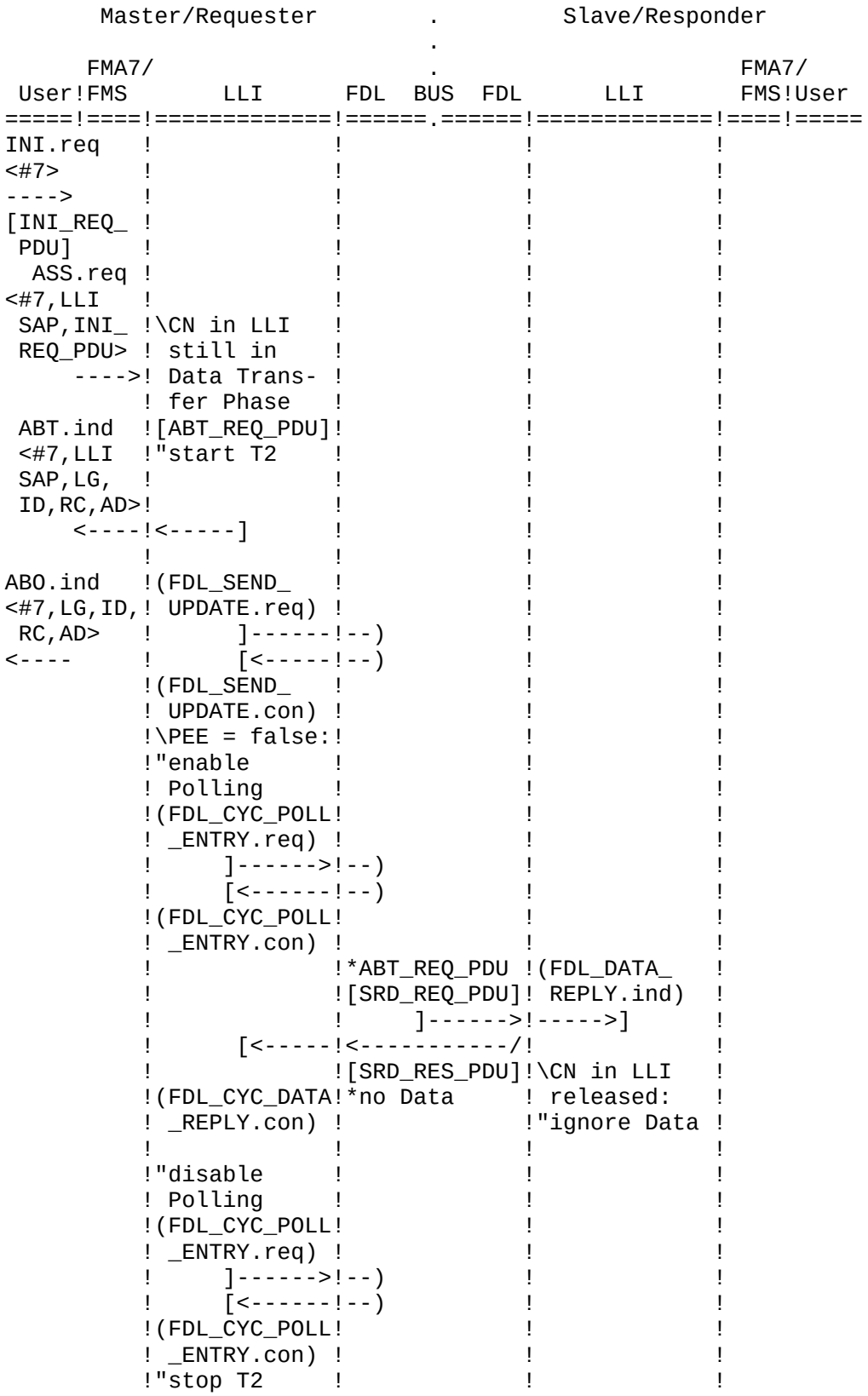


Figure 43. Master-Slave Communication Relationship / all Connection Types / Connection Establishment / Local Error: Connection still established in local LLI

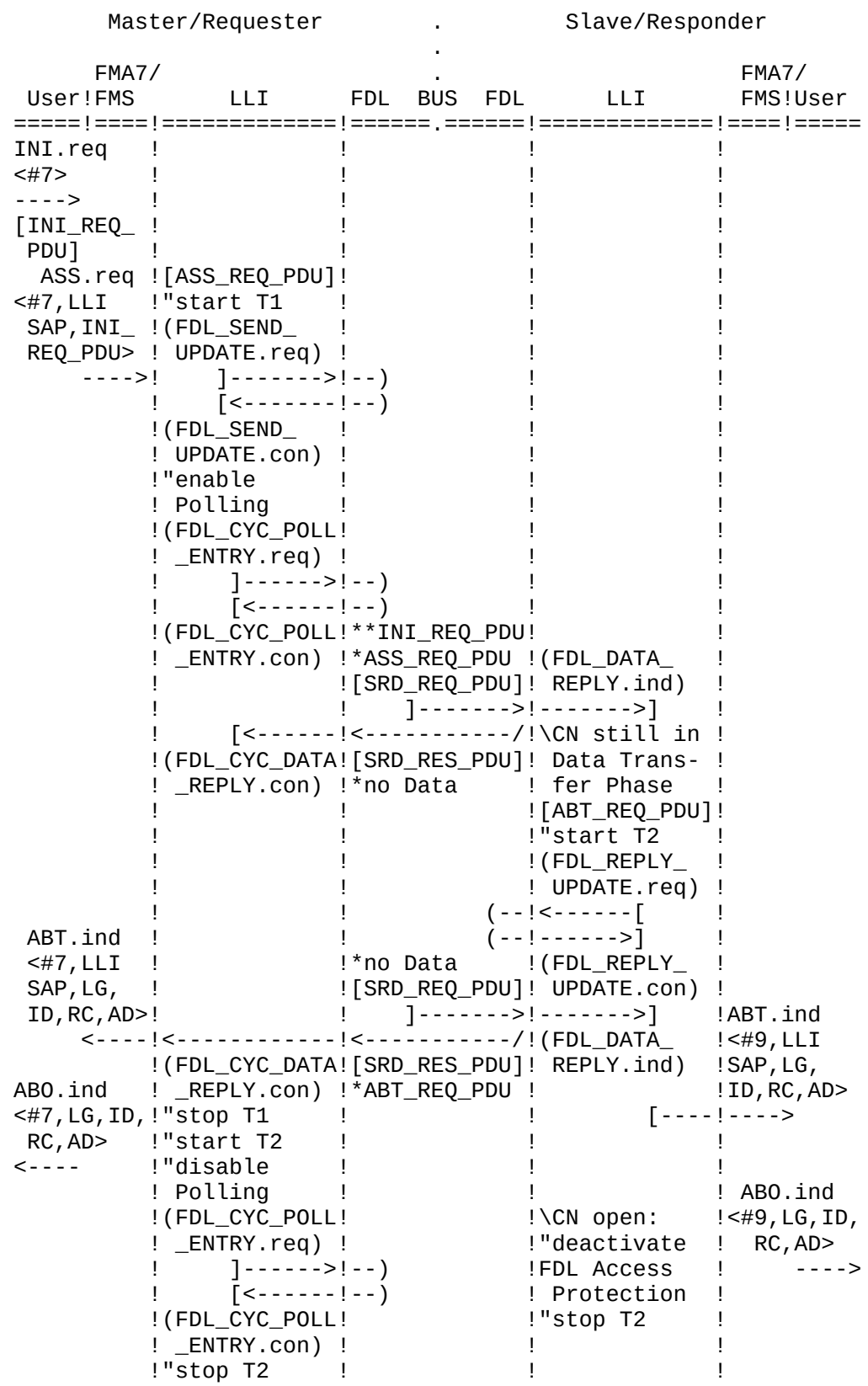


Figure 44. Master-Slave Communication Relationship / all Connection Types / Connection Establishment / Error: Connection still established in remote LLI

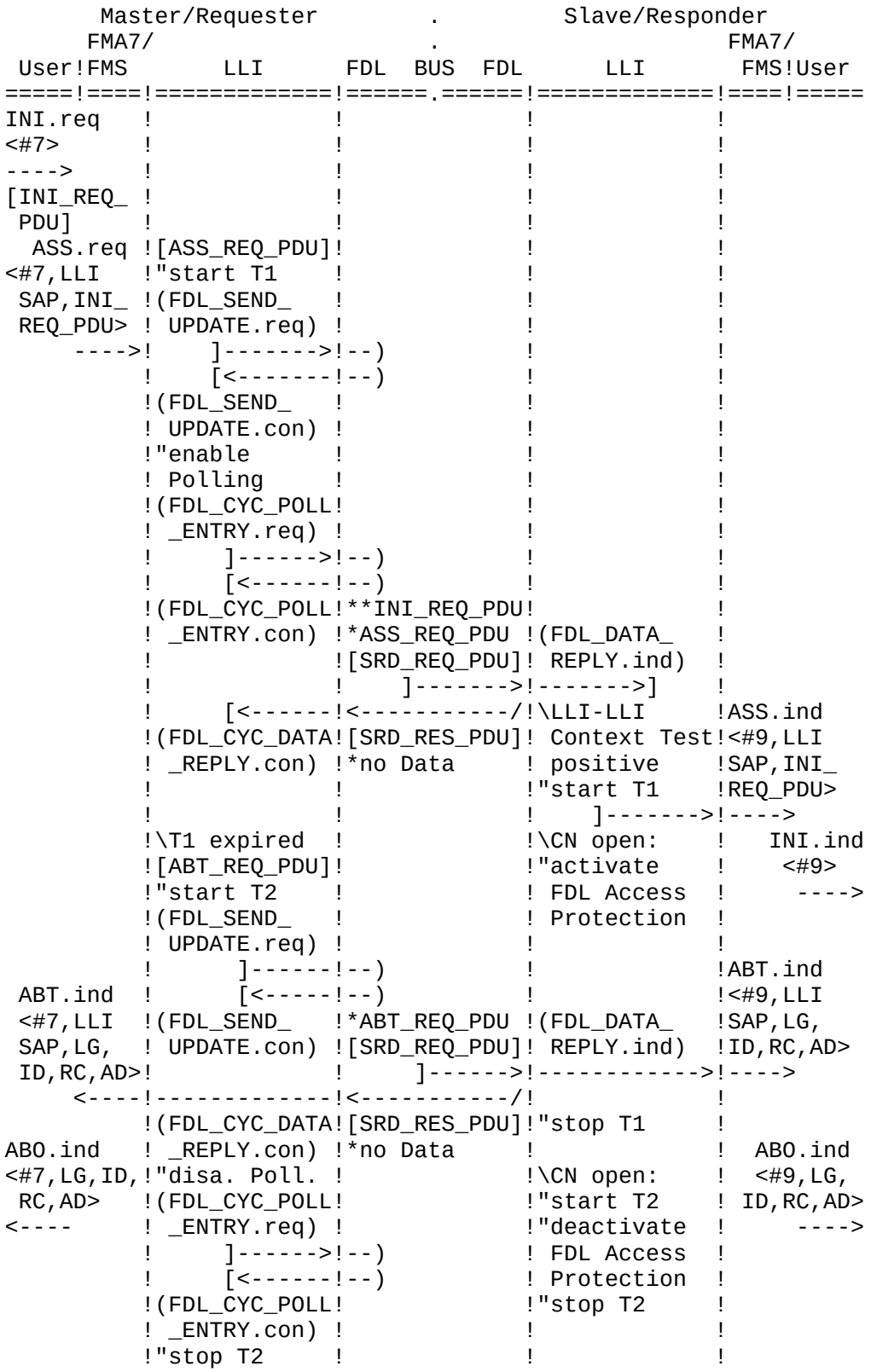


Figure 45. Master-Slave Communication Relationship / all Connection Types / Connection Establishment / Error: T1 expired at the Master

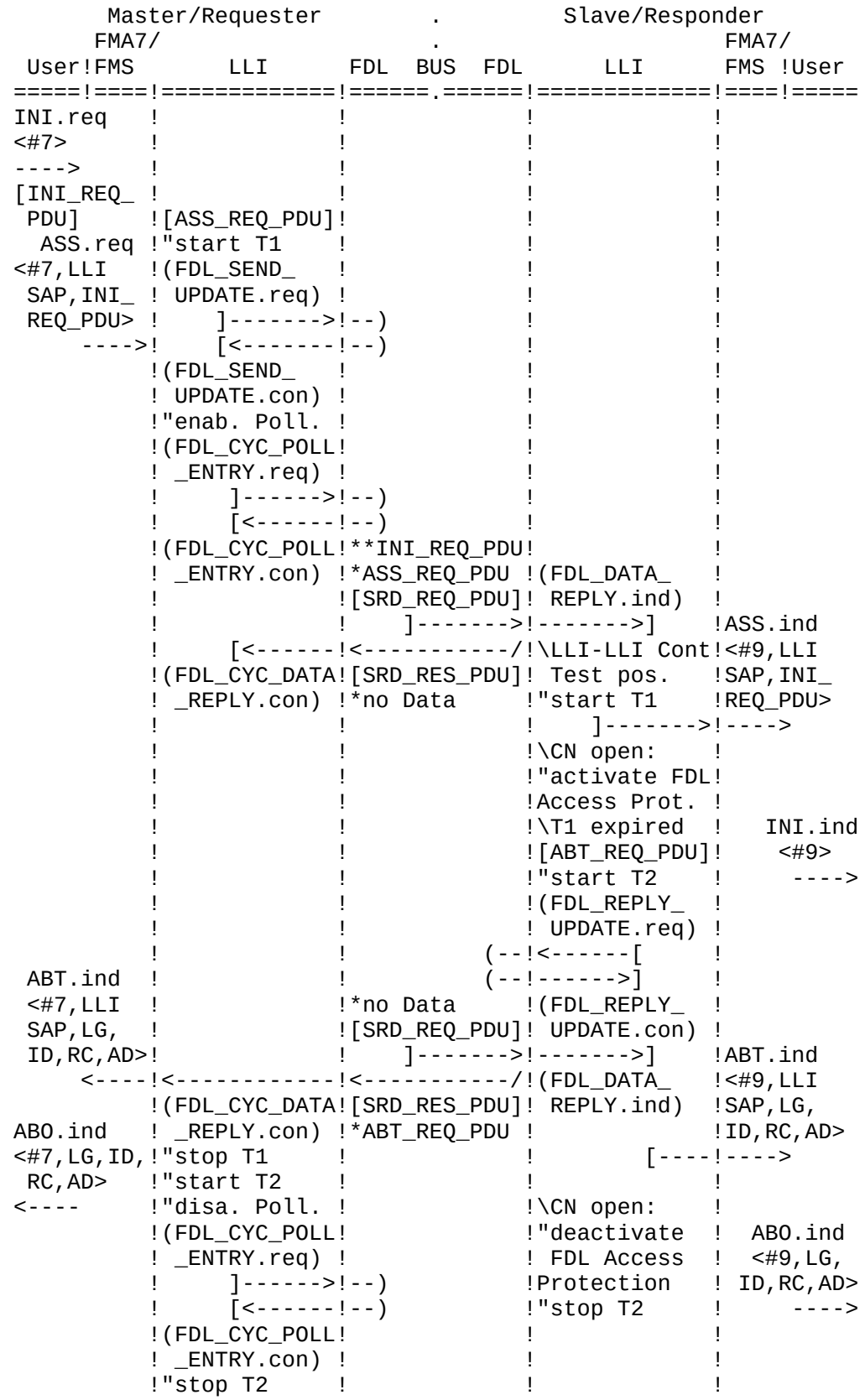


Figure 46. Master-Slave Commun. Relationship / all Connection Types / Connection Establishment / Error: T1 expired at the Slave

6.3.2.3 Associate for Master-Master Communication Relationships

For master-master communication relationships each of the two masters may take the initiative in connection establishment.

If the connection to be established is configured as a master-master communication relationship in the CRL, the Associate service of the LLI is mapped onto the Layer 2 service SDA.

The following sequence charts (see the following eight figures) show examples of possible connection establishment sequences.

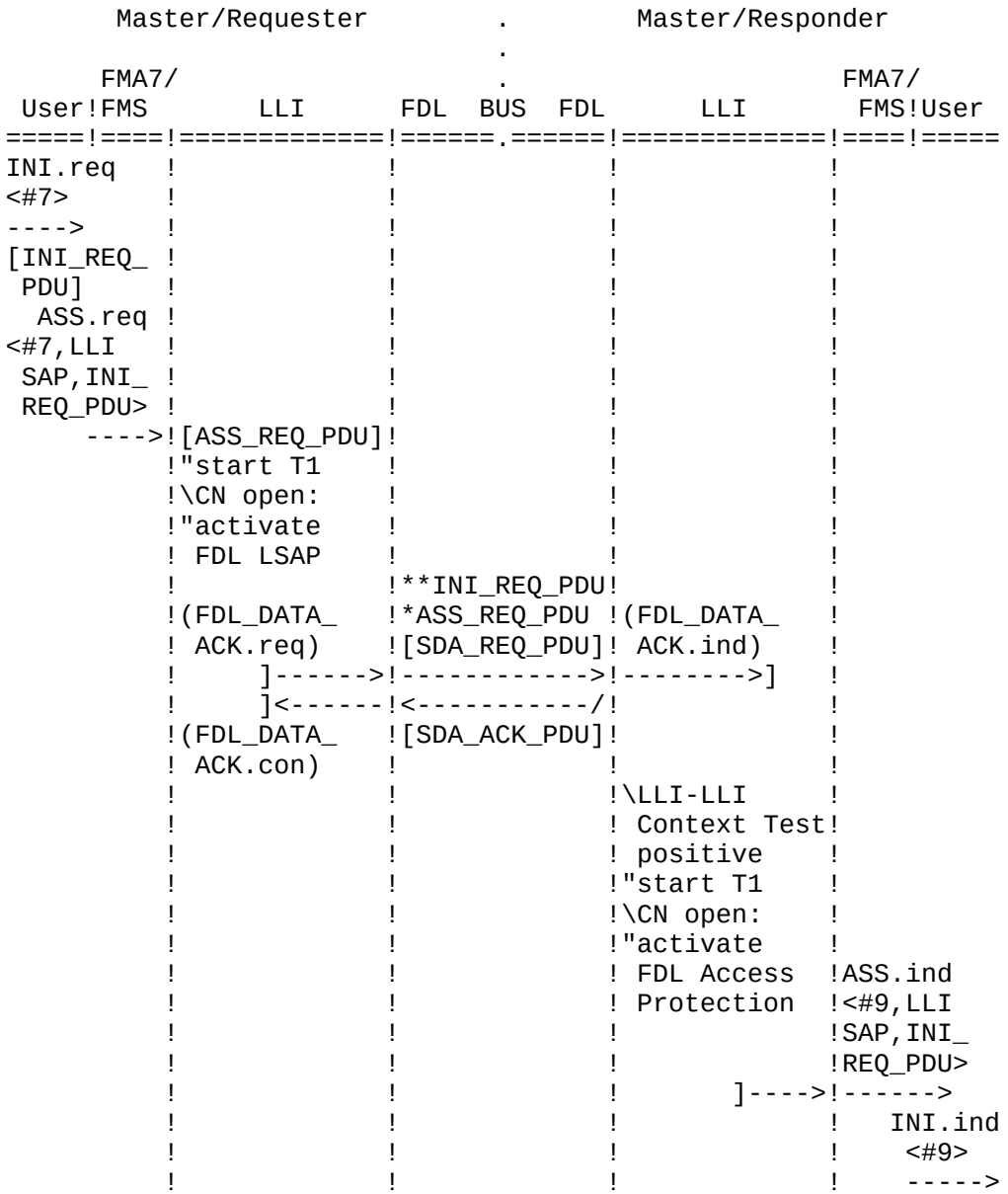


Figure 47. Master-Master Communication Relationship / Connection Establishment / Request / LLI-LLI Context Test positive

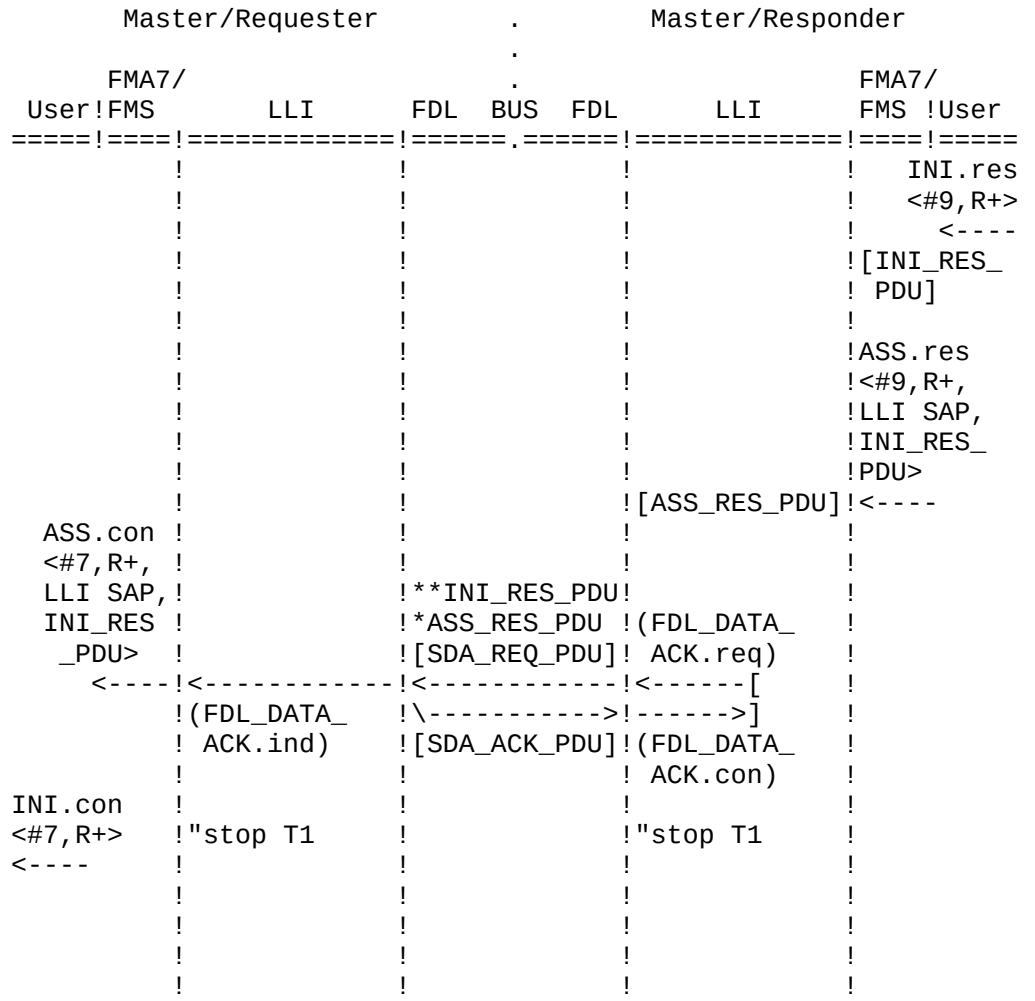


Figure 48. Master-Master Communication Relationship / Connection Establishment / Positive Response

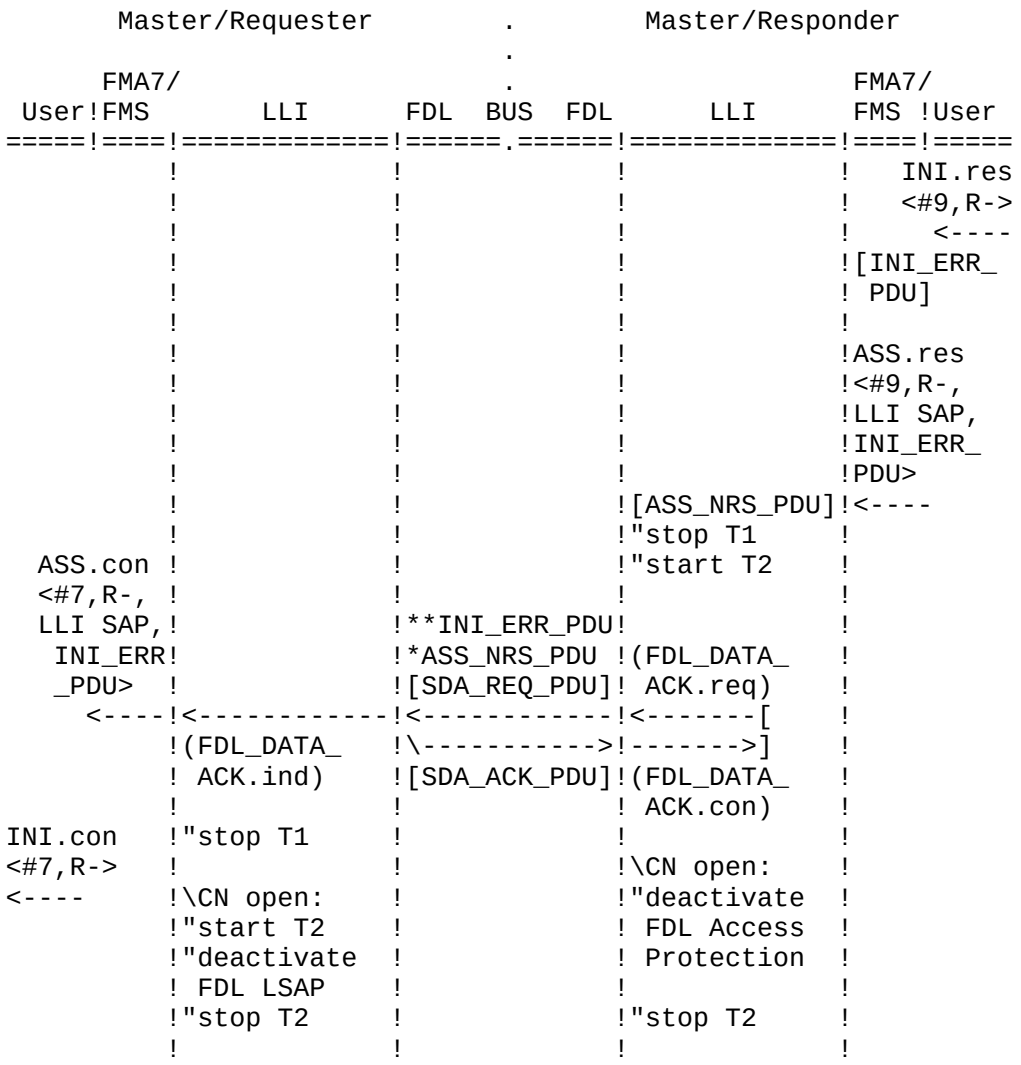


Figure 49. Master-Master Communication Relationship / Connection Establishment / Negative Response

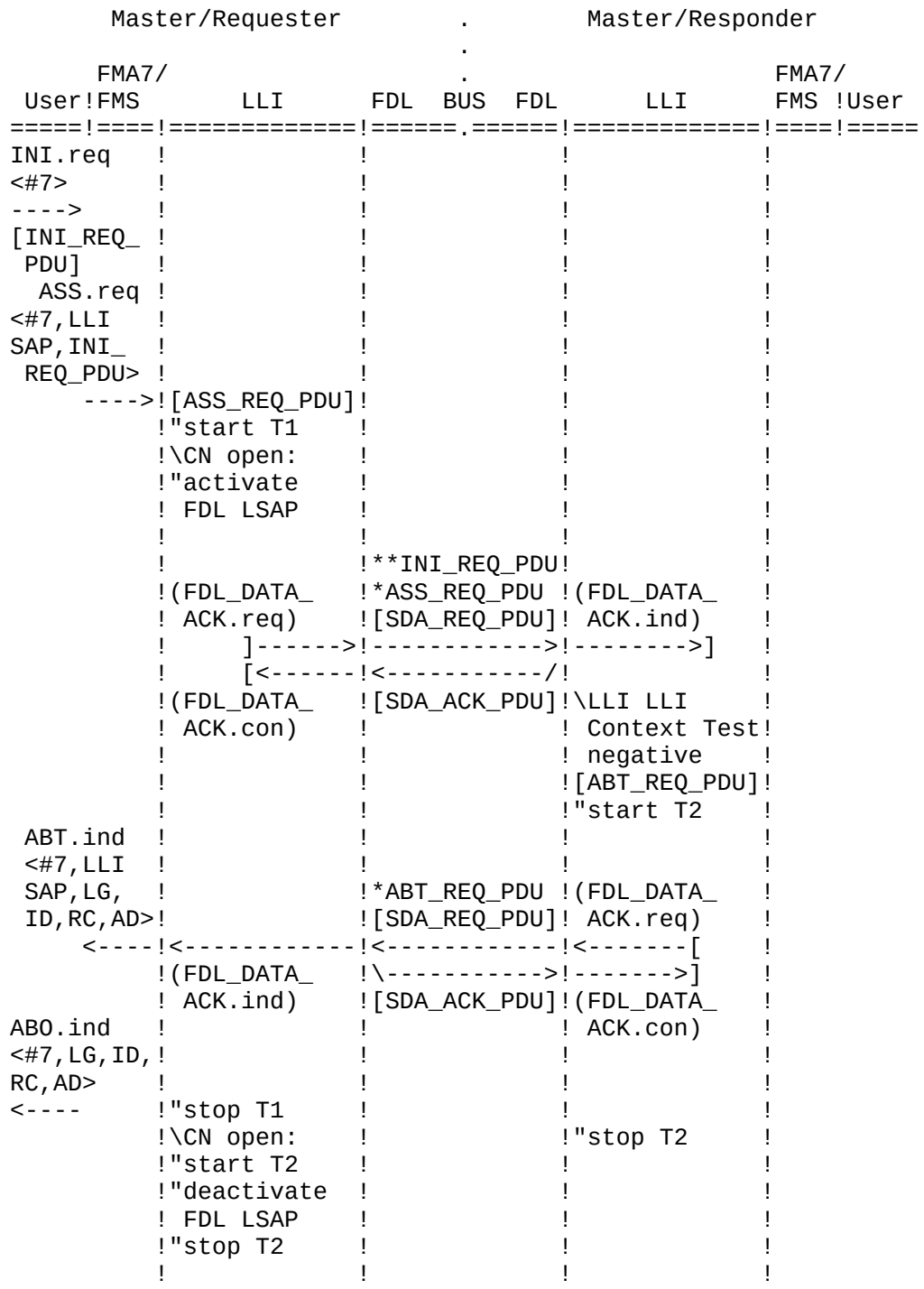


Figure 50. Master-Master Communication Relationship / Connection Establishment / LLI-LLI Context Test negative

The sequence for the local error "CRL-Entry faulty" is the same for master-master and master-slave communication relationships.

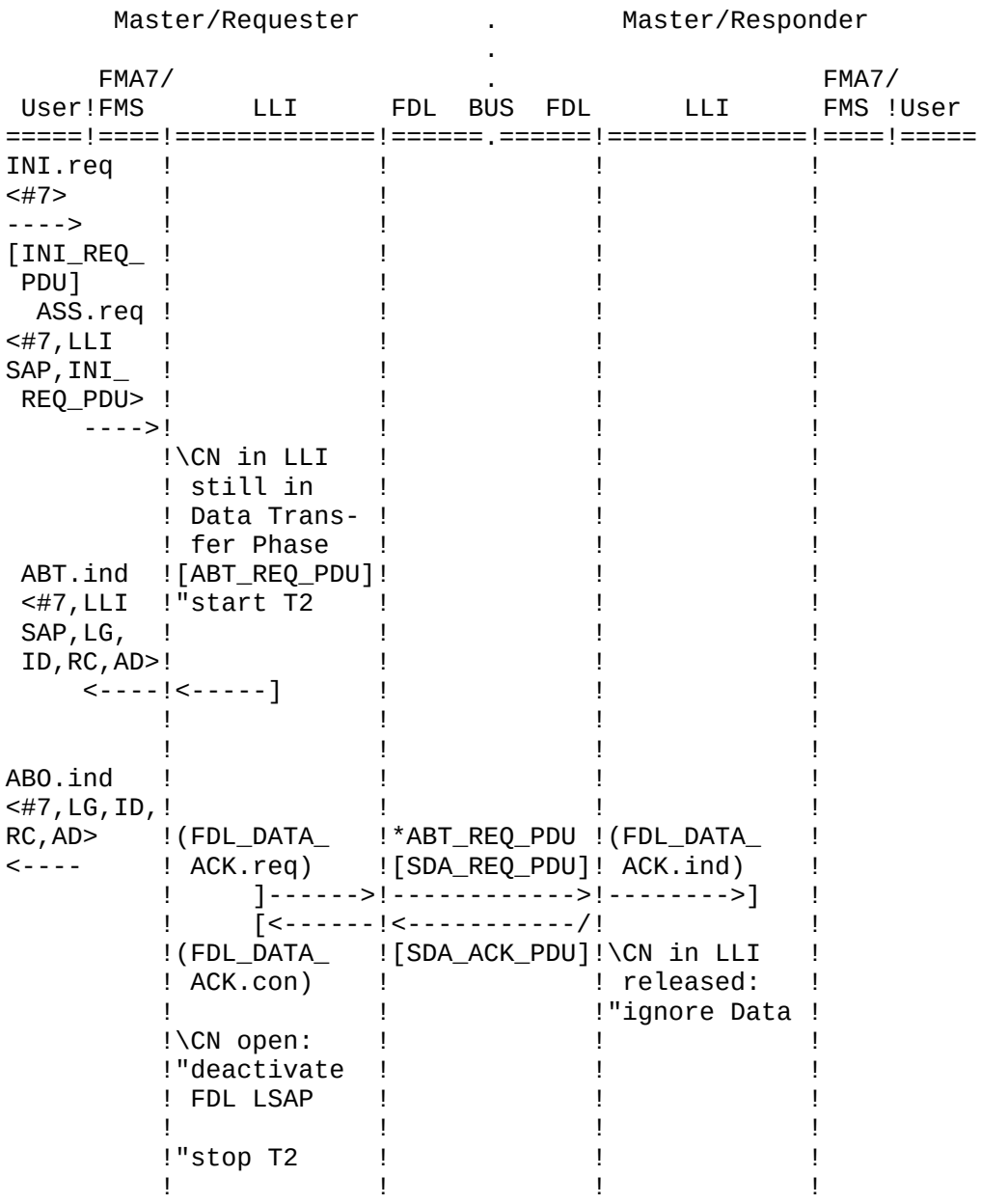


Figure 51. Master-Master Communication Relationship / Connection Establishment / local Error: Connection still established in local LLI

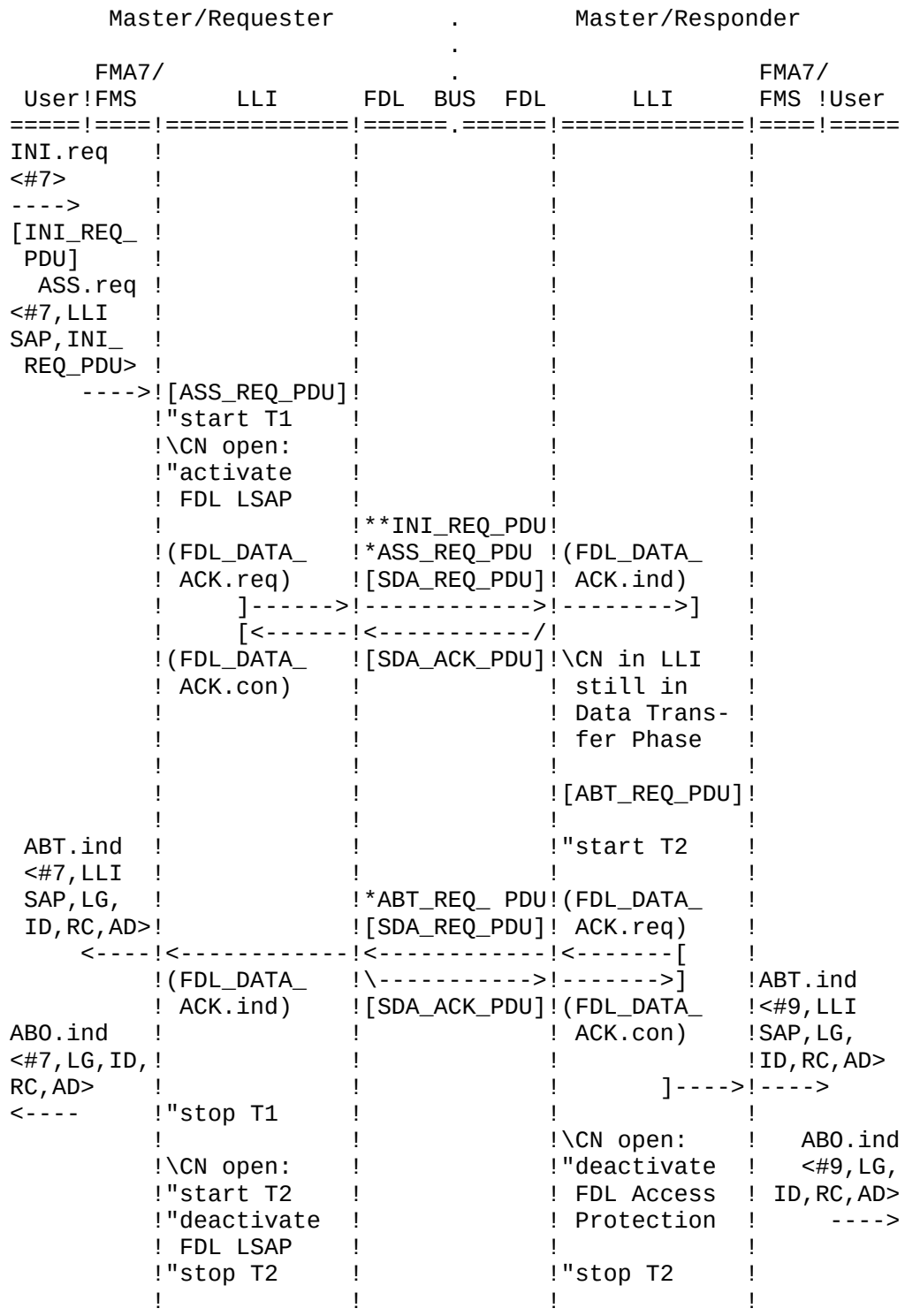


Figure 52. Master-Master Communication Relationship / Connection Establishment / Error: Connection still established in remote LLI

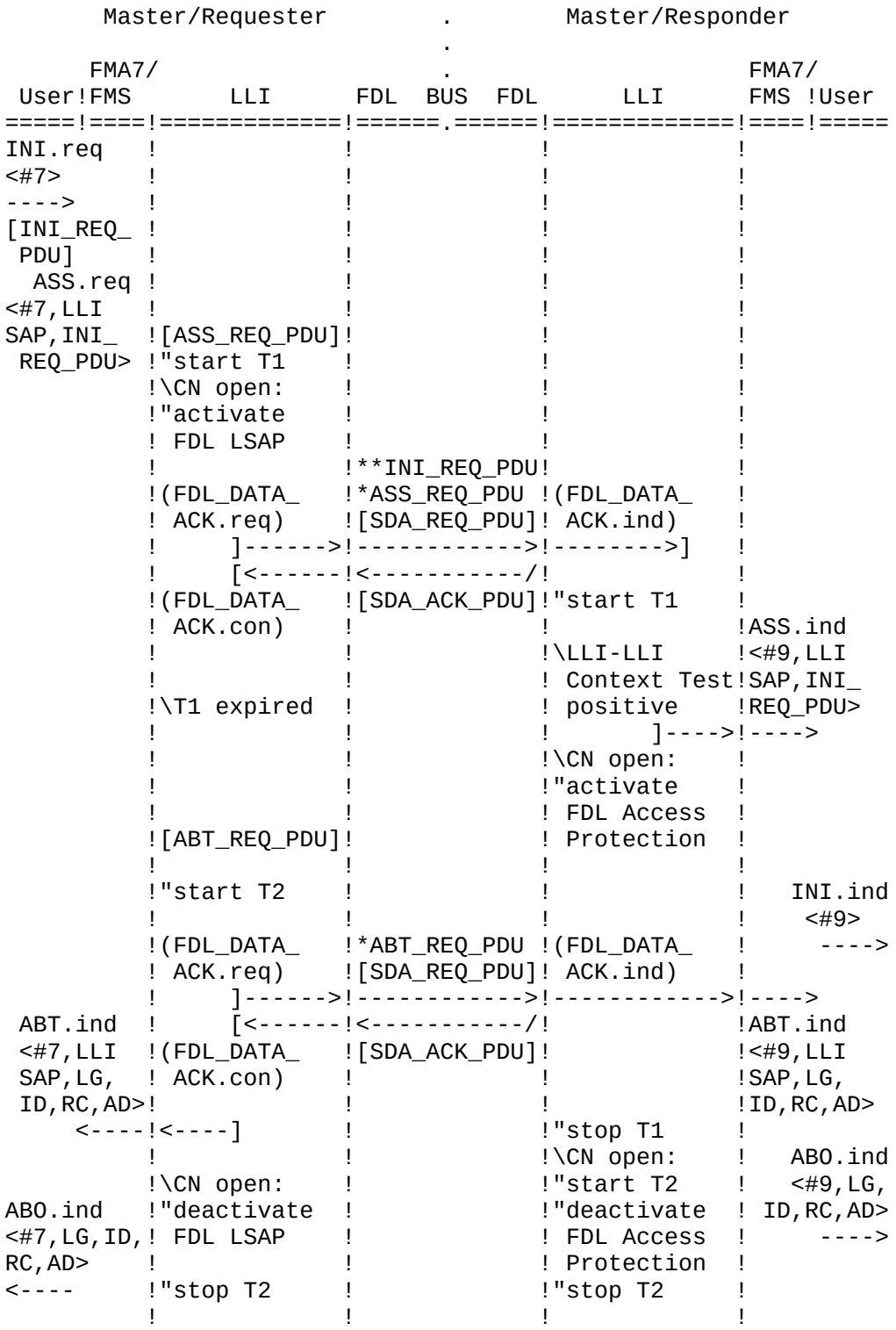


Figure 53. Master-Master Communication Relationship / Connection Establishment / Error: T1 expired at the Re quester

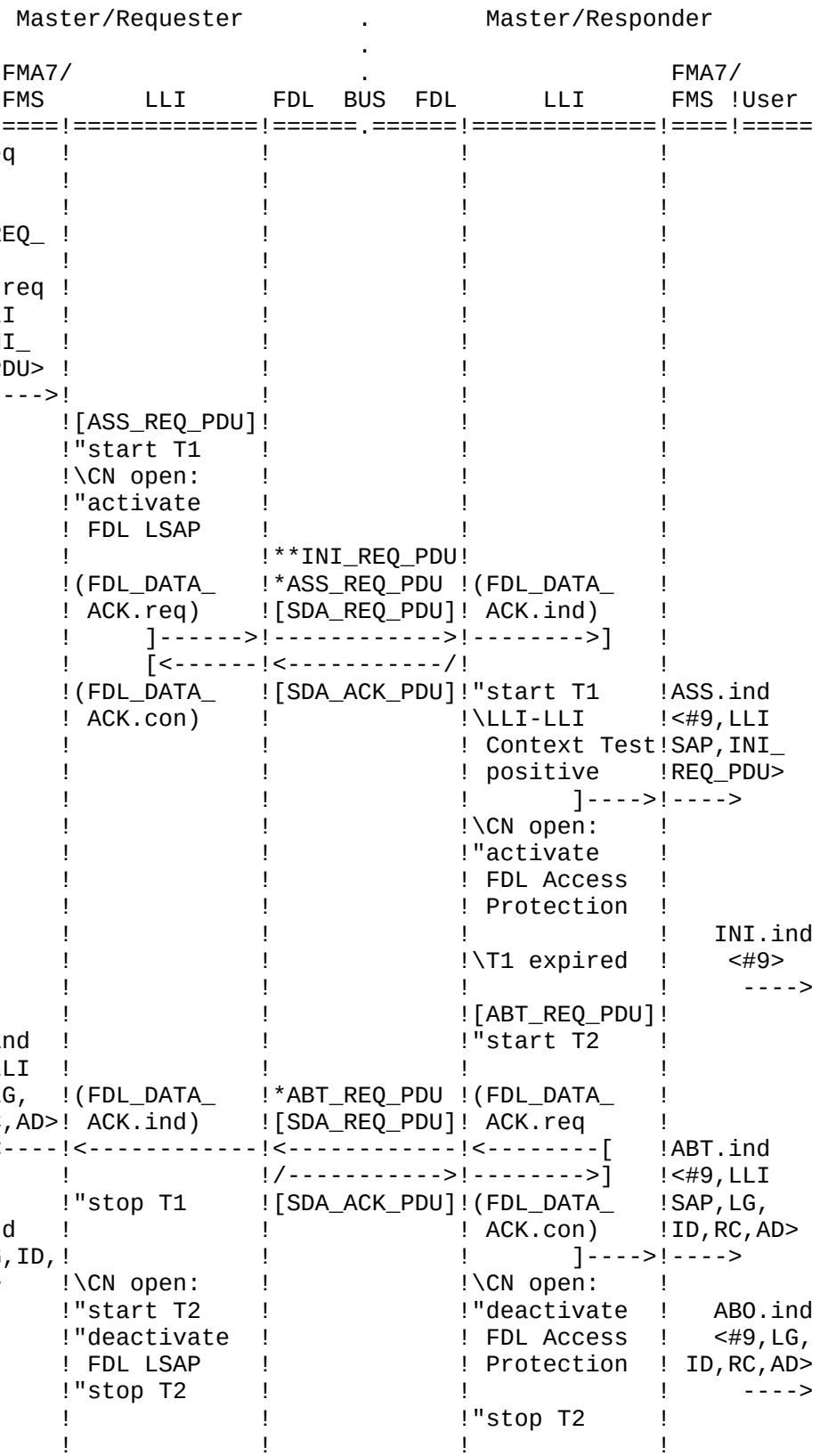


Figure 54. Master-Master Communication Relationship / Connection Establishment / Error: T1 expired at the Responder

6.3.2.4 Handling of Conflicts

During connection establishment on master-master communication relationships it is possible that both masters may try to establish the connection at the same time. This conflict is resolved in the following way:

If the LLI receives an ASS_REQ_PDU of the communication partner before the expected ASS_RES_PDU or ASS_NRS_PDU resp. is received then LLI shall compare the own FDL address (parameter Rem_add of the FDL_DATA_ACK.ind) with the FDL address of the communication partner (parameter Loc_add of the FDL_DATA_ACK.ind). Only the request to establish a connection of that master with the lower FDL address is performed. If the communication partner has the lower FDL address, the own request to establish the connection is rejected with an ABT.ind to the FMS. Thereafter an ASS.ind is passed to the FMS to signal the communication partner's request to establish the connection.

The LLI ignores the communication partner's request to establish the connection if the communication partner has the higher FDL address.

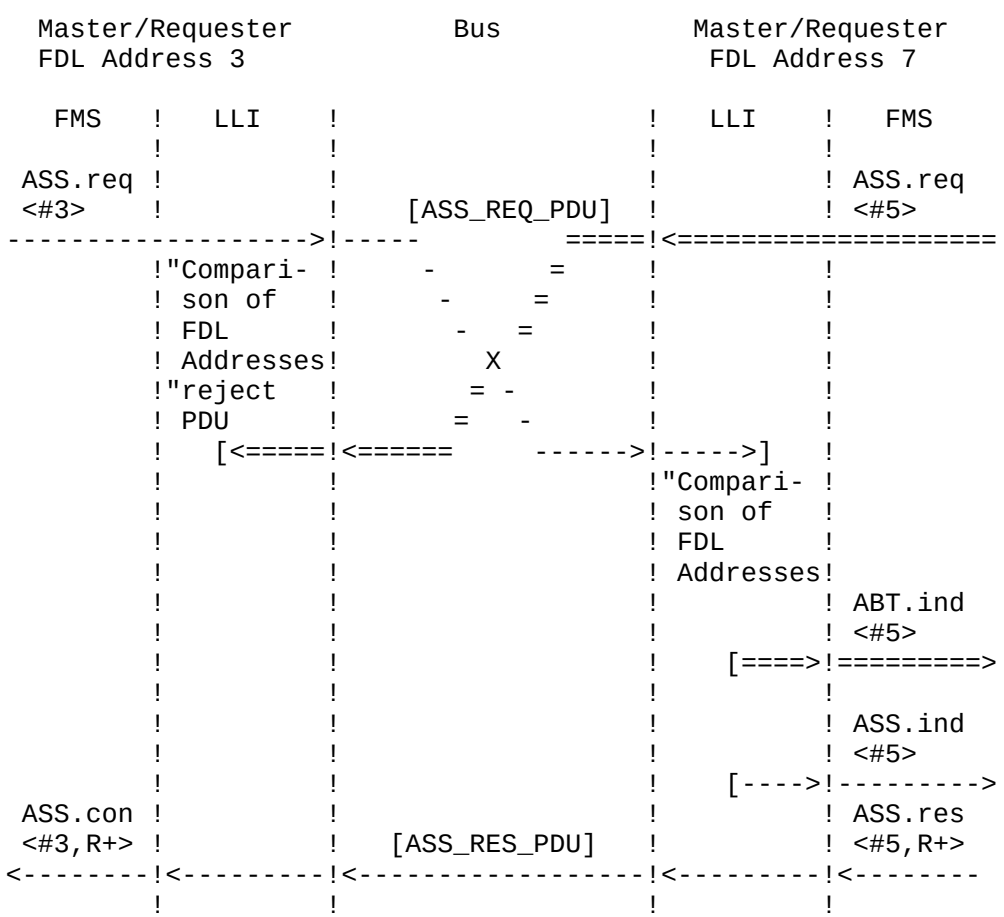


Figure 55. Conflict Resolution for mutual ASS.req, simplified representation

6.3.2.5 Interpretation of the Layer 2 Confirmation Primitive

During the connection establishment phase the parameter L_status of the Layer 2 confirmation primitive (FDL_XXX.con) is interpreted. Certain values of the parameter L_status (see formsl stste mschine definition) also lead to a connection release and in some cases to an error message (LLI-FAULT.ind) to the FMA7.

6.3.2.6 Context Test in LLI

Upon receipt of each ASS_REQ_PDU the LLI of the responder checks for the connection to be established that the LLI context of the communication partner (remote context) is compatible with the own LLI context (local context) in the CRL. This is called LLI-LLI context test.

The local context is assumed to be correctly configured. The same rule shall apply to the local context between LLI and FMS.

The compatibility of the remote to the local context is shown in the following matrix:

Table 19. Compatibility of the local to remote Context

remote Context	local Context					
	TYPE	max SCC	max RCC	max SAC	max RAC	ACI
TYPE	=					
max SCC		•				
max RCC			•			
max SAC				•		
max RAC					•	
ACI						=

Explanation:
≤ : local value smaller than or equal remote value
≥ : local value larger than or equal remote value
= : local value equal to remote value

6.3.3 Connection Release

The connection release closes an existing logical connection between two communication partners. A connection may be released in the master or slave by user initiative, or in error cases by initiative of FMS, FMA7 or LLI.

The connection release is performed using the unconfirmed LLI service Abort (ABT). A released connection shall be established again using the LLI service Associate (Ass) before it may be used again for data transfer. If a connection is in the state "CLOSED" (released), all LLI service requests with exception of the Associate service are rejected locally with an ABT.ind. All received PDUs from the remote station with exception of the ASS_REQ_PDU or ABT_REQ_PDU resp. are rejected with an ABT_REQ_PDU in this state.

If the FMS user wishes to release an existing connection it passes the request to release the connection (Abort.req) to the FMS. If the FMA7 user wishes to release an existing connection it passes the request to release the connection (FMA7-Abort.req) to FMA7. FMS and FMA7 pass the request to release the connection of their users with an ABT.req service primitive to LLI.

If the local FMS detects errors it passes an Abort.ind to the user and the service primitive ABT.req to LLI.

If the local FMA7 detects errors it passes an FMA7-Abort.ind to the user and the service primitive ABT.req to LLI.

If the local LLI detects errors (e.g. reaction of the LLI connection) it passes an ABT.ind to the LLI user. The LLI user passes this ABT.ind to the user.

In all cases the LLI of the requester starts the monitoring of connection release (T2, see connection release definition) and generates an ABT_REQ_PDU. The LLI enters the reason for the connection release into the fields ID, RC and AD. The LLI passes the ABT_REQ_PDU to Layer 2 for transmission to the remote communication partner. After transmission the connection is in the state "CLOSED" (released). In the case of an open connection in the responder (Connection Attribute = "O") the LLI shall stop the access protection for the assigned LSAP after the connection release. Thereafter the access for all remote partners is possible again. In the case of an open connection in the requester (Connection Attribute = "I") the LLI shall deactivate the assigned LSAP after the connection release.

After the receipt of an ABT_REQ_PDU the LLI of the receiver performs a local connection release. If local actions of Layer 2 are necessary for this connection release (e.g. deactivation of an LSAP), then the LLI starts the monitoring of connection release.

The reason for the connection release is found in the fields ID, RC and AD of the ABT_REQ_PDU and is passed to the LLI user with the ABT.ind. The FMS passes this ABT.ind to the user. Thereafter the connection is in the state "CLOSED" (released). In the case of an open connection in the responder (Connection Attribute = "O") the LLI shall stop the access protection for the assigned LSAP after the connection release. In the case of an open connection in the requester (Connection Attribute = "I") the LLI shall deactivate the assigned LSAP after the connection release and thus re-store the initial state for a new request to establish a connection. Then the monitoring of connection release is stopped if necessary.

6.3.3.1 Monitoring of Connection Release

For time monitoring of connection release the timer T2 is used. It is connection specific and mandatory for master and slave devices. The interval for monitoring connection release is configured in the header of the LLI CRL for all connections.

In the requester the timer T2 controls the sending of the ABT_REQ_PDU. The timer T2 is started before the ABT_REQ_PDU is passed to Layer 2 and is stopped after receiving the Layer 2 confirmation for sending (FDL_XXX.con/FDL_XXX.ind) and after the termination of all local actions.

In the receiver of an ABT_REQ_PDU the timer T2 controls the connection release if local actions in Layer 2 (e.g. stopping of Layer 2 polling) or in FMA1/2 (e.g. deactivation of LSAPs) are necessary. The timer T2 is started upon receipt of the ABT_REQ_PDU and is stopped after termination of all local actions.

If the timer T2 expires the connection release is stopped and the connection changes into the state "CLOSED". If a local Layer 2 error has occurred, then the FMA7 is informed additionally.

6.3.3.2 Abort for Master-Slave Communication Relationships

For master-slave communication relationships the master as well as the slave may take the initiative for connection release. Besides the general rules which are described in connection release the following specific rules for this communication relationship are valid.

The Abort service (ABT) of the LLI is mapped onto the cyclic Layer 2 service CSRD.

If the Layer 2 polling to the assigned slave (Rem_add/DSAP) is locked in the master of a connection for Acyclic Data Transfer with no Slave Initiative, the LLI shall start the polling for the transmission of the ABT_REQ_PDU. Thereby the Layer 2 service primitive FDL_CYC_POLL_ENTRY.req with parameter Marker = "unlock" is used. If the ABT_REQ_PDU has been transmitted or timer T2 expires, then the LLI stops the polling to the assigned slave for all master-slave connections. Thereby the Layer 2 service primitive FDL_CYC_POLL_ENTRY.req with parameter Marker = "lock" is used.

If the master receives an ABT_REQ_PDU then the LLI shall stop the Layer 2 polling to the assigned slave.

The transmission of the ABT_REQ_PDU from the slave to the master is only guaranteed if the slave is still being polled by the master's CSRD service.

EXAMPLE: The following sequence charts (see the following five figures) show examples of possible connection release sequences.

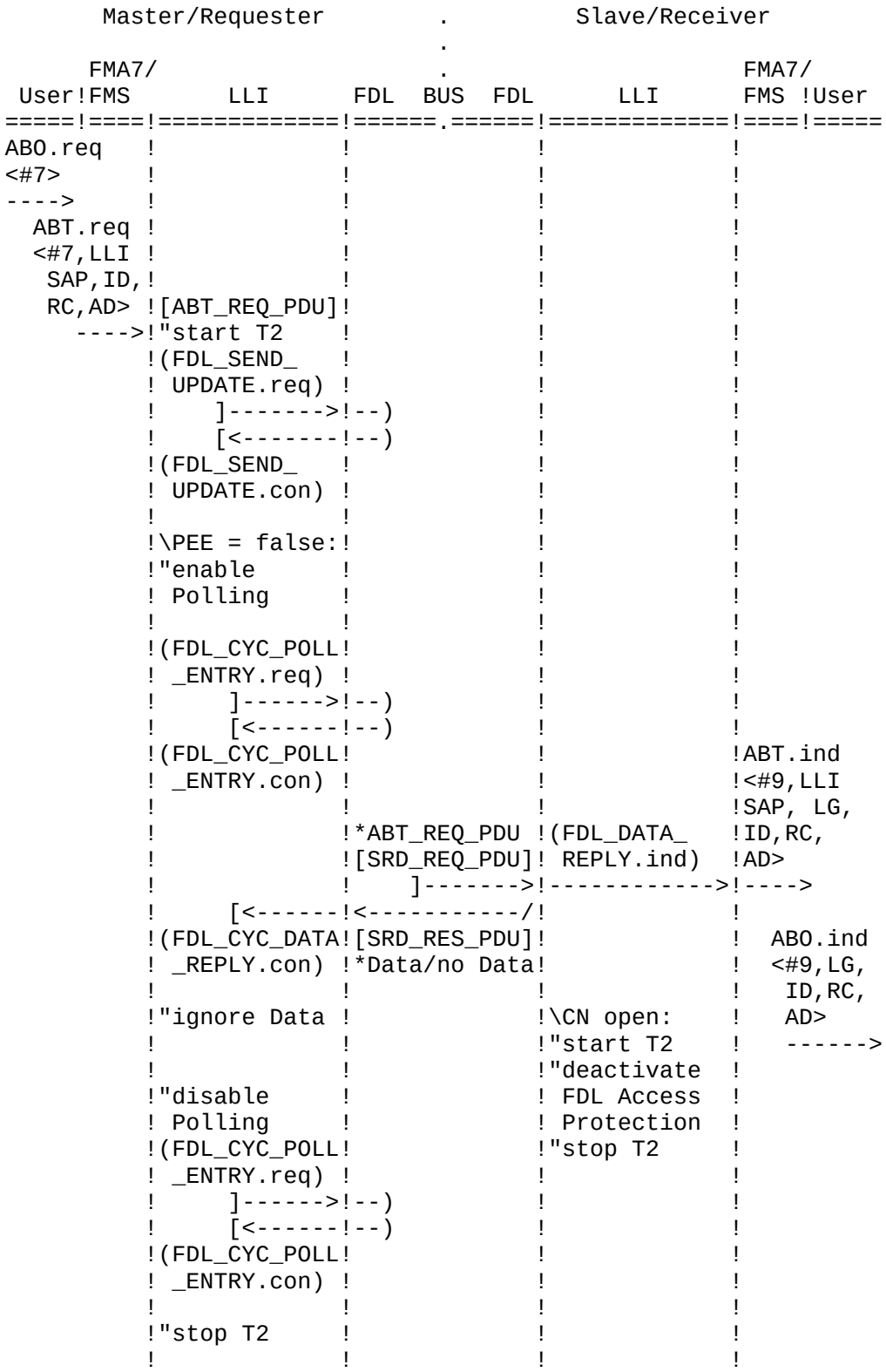


Figure 56. Master-Slave Communication Relationship / Connection for Acyclic Data Transfer with no Slave Initiative / Connection Release by the User of the Master

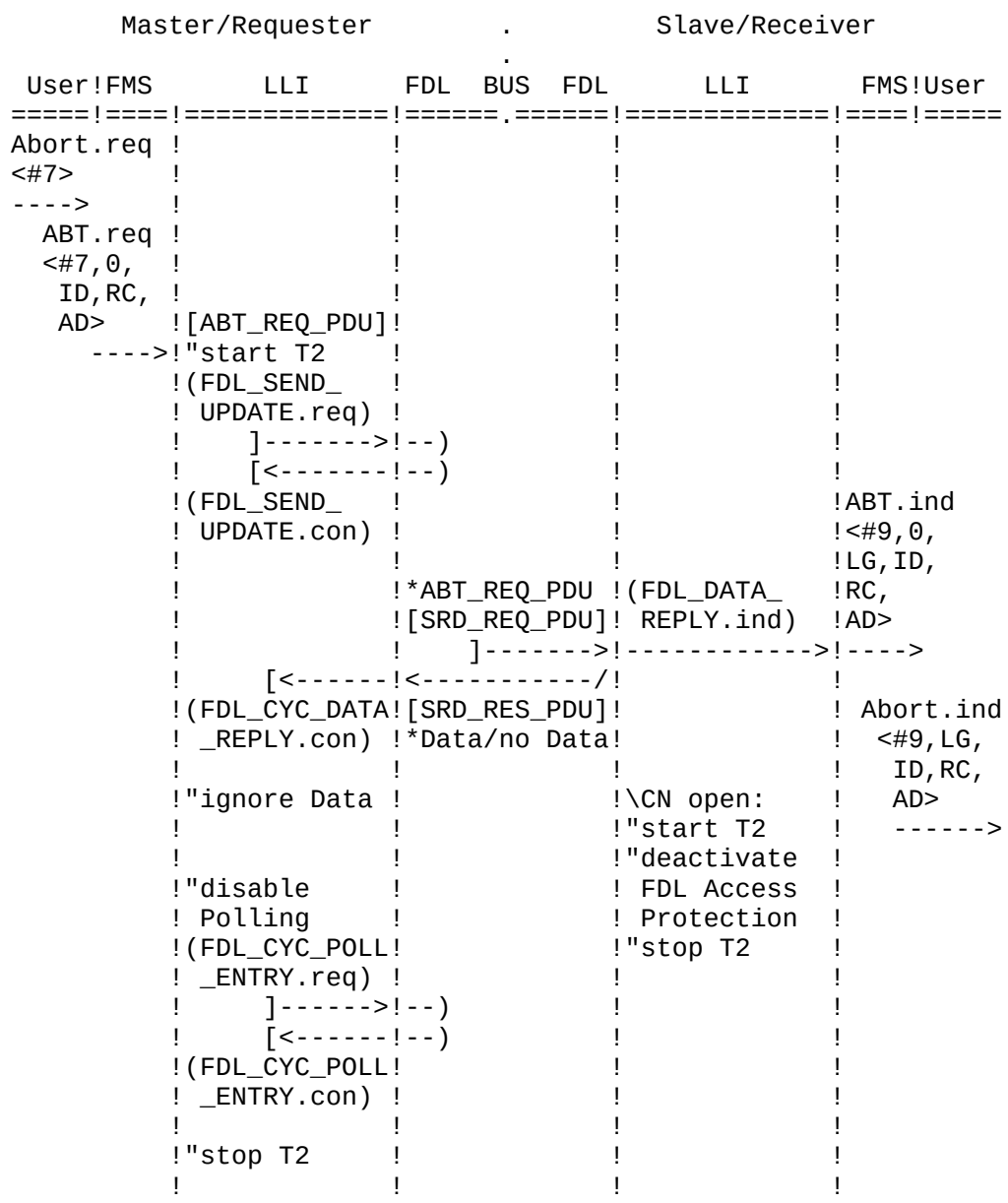


Figure 57. Master-Slave Communication Relationship / Connection for Acyclic Data Transfer with Slave Initiative or Connection for Cyclic Data Transfer / Connection Release by the User of the Master

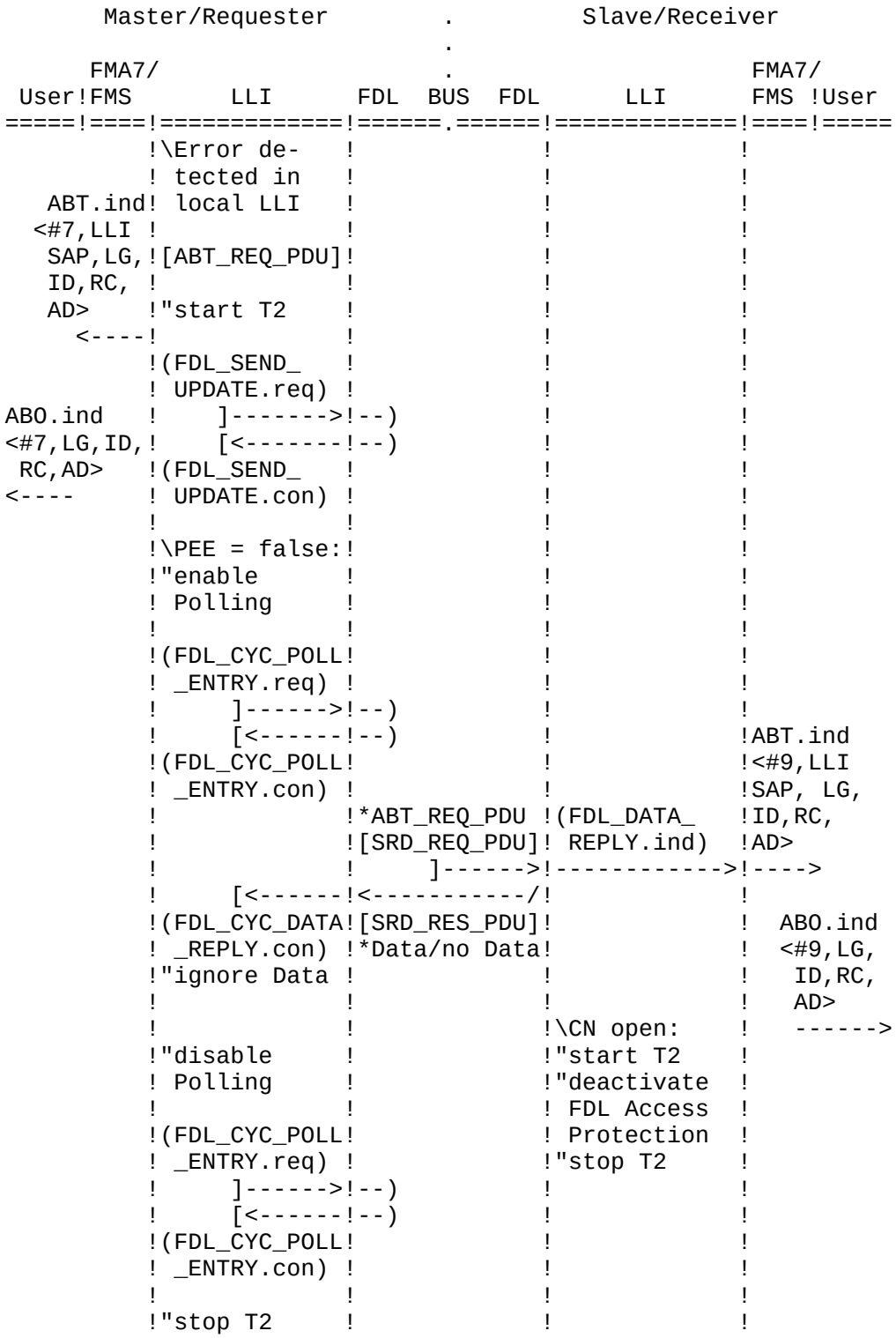


Figure 58. Master-Slave Communication Relationship / Connection for Acyclic Data Transfer with no Slave Initiative / Connection Release / Error detected in LLI at the Master

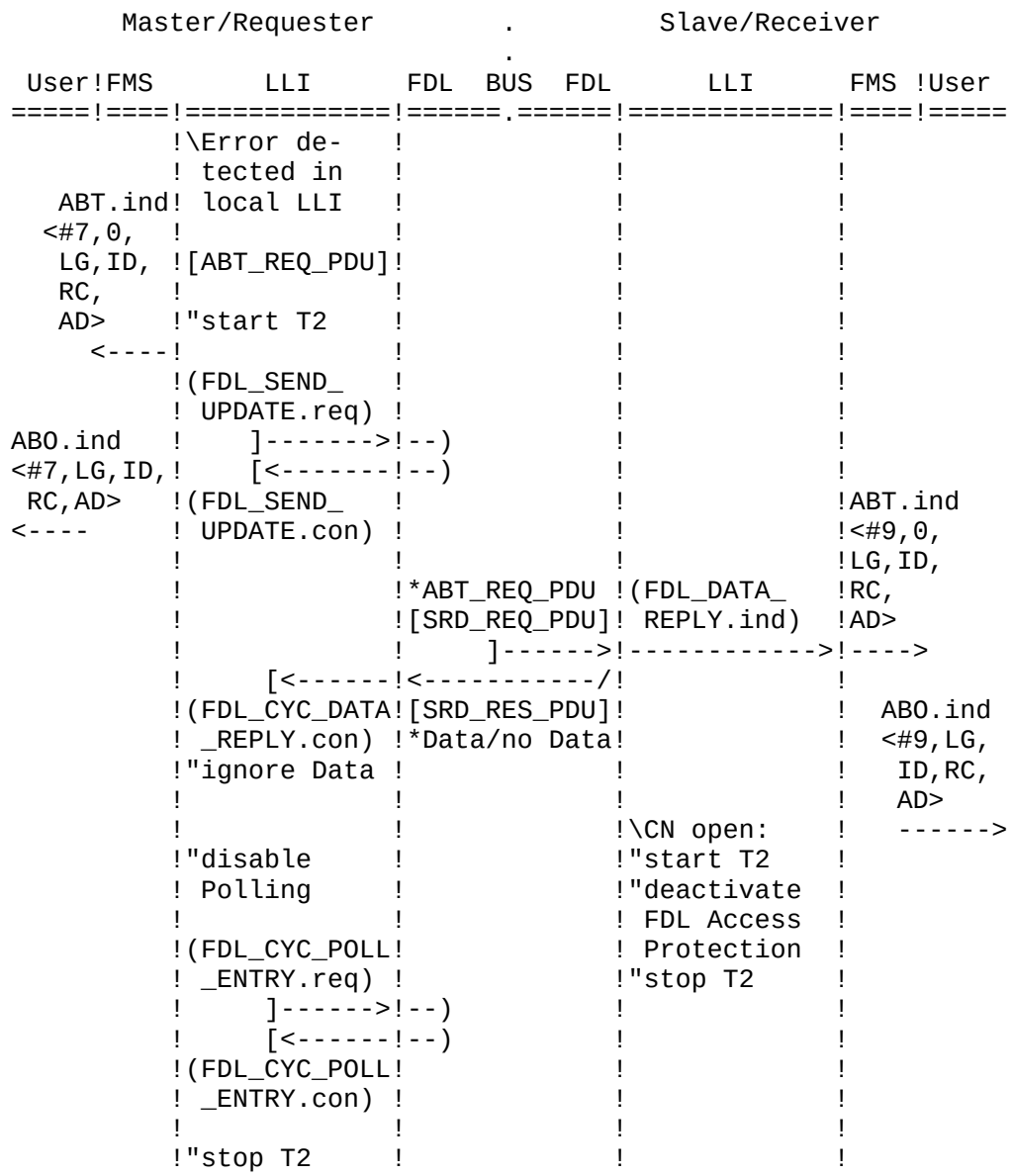


Figure 59. Master-Slave Communication Connection for Acyclic Data Transfer with Slave Initiative or Connection for Cyclic Data Transfer / Connection Release / Error detected in LLI at the Master

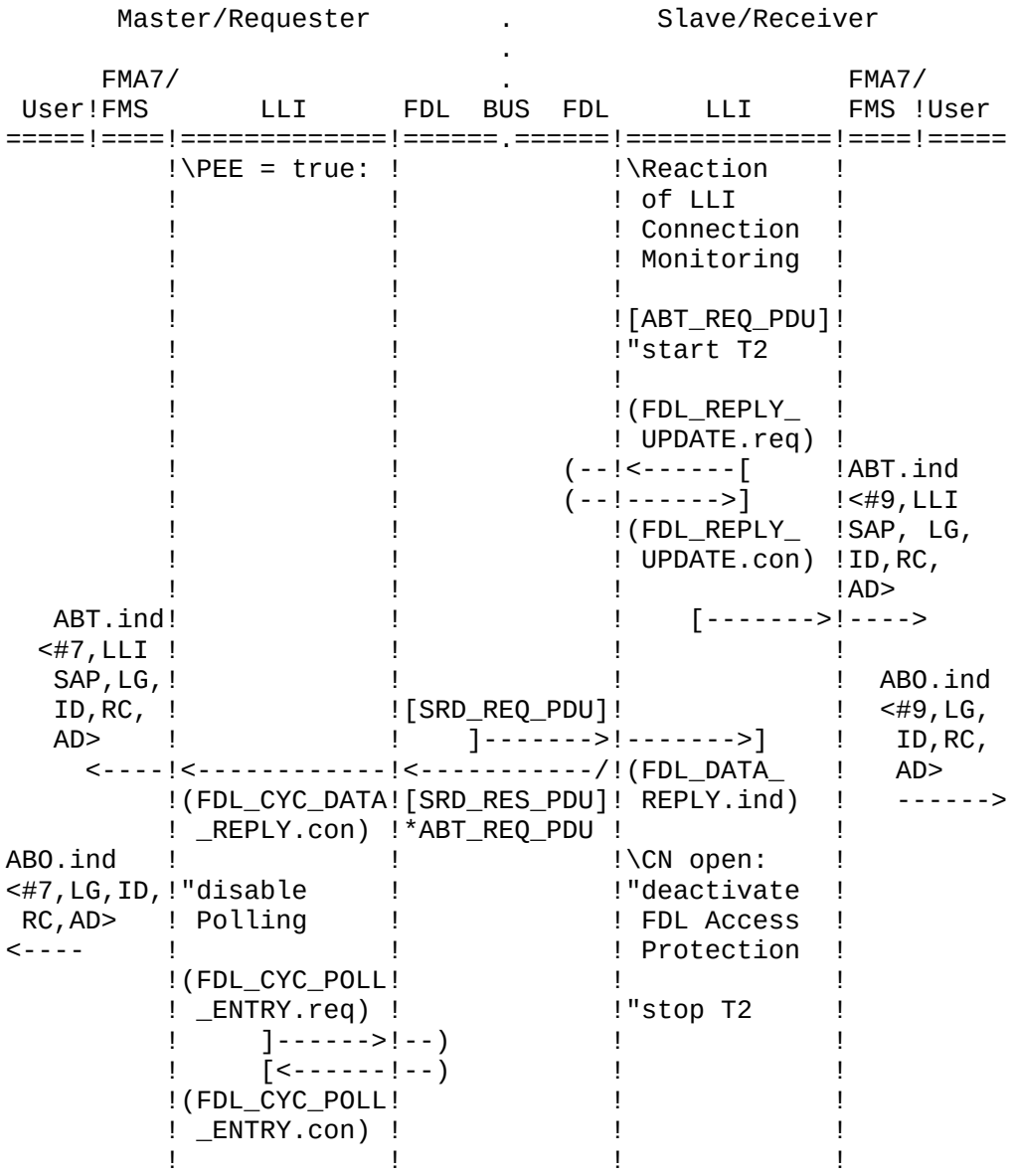


Figure 60. Master-Slave Communication Relationship / all Connection Types / Connection Establishment / Error detected in LLI at the Slave

6.3.3.3 Abort for Master-Master Communication Relationships

If the communication relationship to be released is configured in the CRL as a master - master connection, then the Abort service (ABT) of the LLI is mapped onto the Layer 2 service SDA.

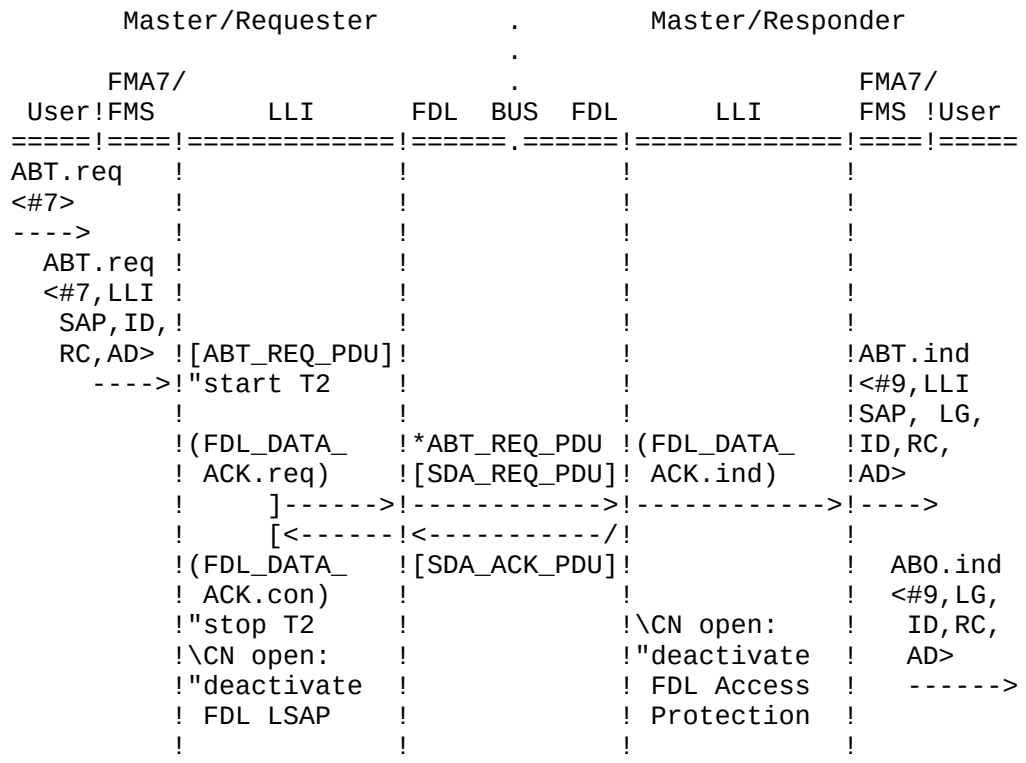


Figure 61. Master-Master Communication Relationship / Connection Release of the User

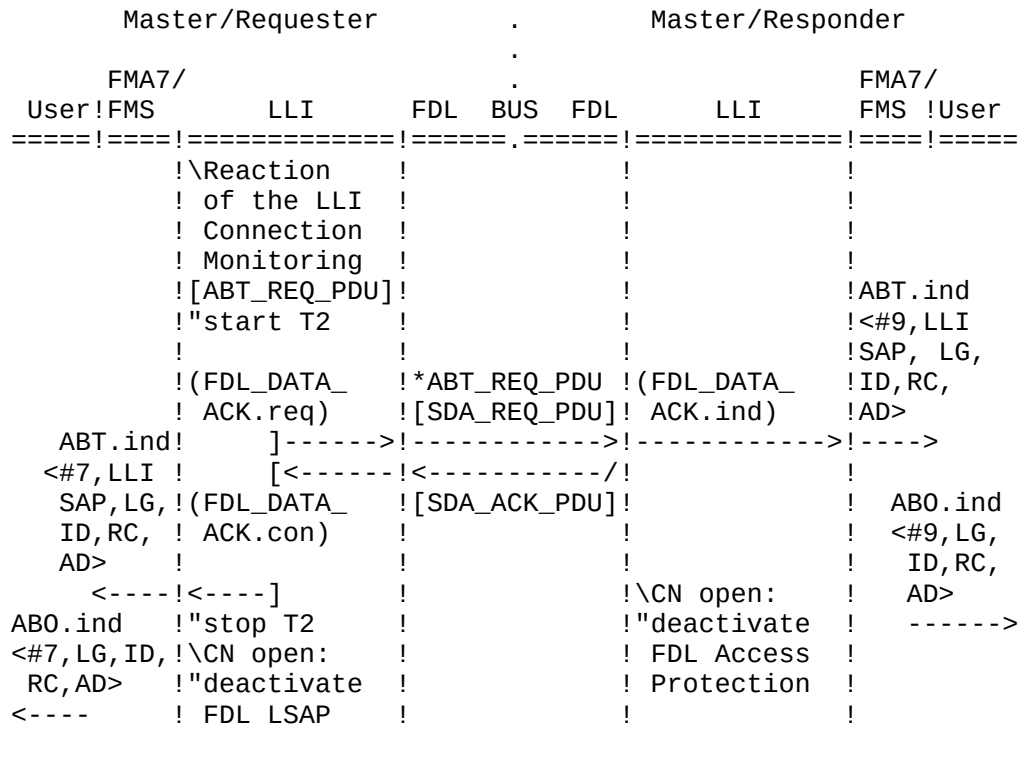


Figure 62. Master-Master Communication Relationship / Connection Release / LLI Connection Monitoring

6.3.3.4 Interpretation of the Layer 2 Confirmation Primitive

During the connection release phase the parameter L_status of the Layer 2 confirmation primitive (FDL_XXX.con) is interpreted. Certain values of the parameter L_status (see formal LLI state machine) also lead to an error message (LLI-FAULT.ind) to FMA7.

6.3.4 Data Transfer

6.3.4.1 Mapping of FMS/FMA7 Services onto Layer 2 for a Master-Slave Communication Relationship

This subclause describes the mapping of the FMS/FMA7 services onto the Layer 2 services. A master-slave communication in Layer 2 is assumed here. The Layer 2 services CSRD and SRD are used. The advantage of these services lies in the fast communication by using the immediate response feature of the PROFIBUS FDL protocol. All master-slave communication relationships use the common Poll List SAP in the master.

All master-slave connections use a common LSAP in the master. This LSAP (Poll List LSAP) contains the Poll List of Layer 2.

If the service primitives FDL_SEND_UPDATE.req or FDL_REPLY_UPDATE.req are called, the parameter Transmit shall have the value "single". This ensures that an update is transmitted once only.

6.3.4.2 Connection for Cyclic Data Transfer with no Slave Initiative (MSCY)

Connections for Cyclic Data Transfer with no Slave Initiative shall be established like all other connection oriented communication relationships using a connection establishment service (Initiate). During the connection establishment the polling is started for this communication relationship (FDL_CYC_POLL_ENTRY.req). From that time on the slave is polled with SRD_REQ_PDUs corresponding to its entries in the Poll List. Multiple entries are possible to shorten the reaction time.

During the following data transfer phase only the confirmed FMS services Read and Write and all unconfirmed FMS services are allowed. In this case the master is always requester and the slave responder or receiver. The LLI maps the Reject service of FMS onto the Abort service. The transmission of remote FMA7 services is not allowed on connections for Cyclic Data Transfer with no Slave Initiative.

The LLI of the master controls the actualization of data in the slave (REPLY_UPDATE) and thereby the connection to the slave. Optionally the slave may monitor the connection to the master (see data transfer definition).

The LLI of the requester performs a confirmed FMS service cyclically until a new confirmed FMS service (with a new Invoke ID) is passed to the LLI or an Abort releases the connection.

Read service:

On a connection for Cyclic Data Transfer with no Slave Initiative the Read service is used for cyclic reading of data from a slave. The read data received is stored specific to the connection in the Image Data Memory (IDM) of the master. For each connection only one read request (Read.req) is permissible at the same time. The user marks each read request with a special identification (Invoke ID). In this way it is possible to relate read requests, which have been sent, to read confirmations (Read.con), which have been received. The FMS maps a read request (Read.req) from the user onto the LLI service primitive DTC.req. The Invoke ID is stored in the IDM in the LLI of the master. Only the first or a changed read request causes a transmission of the READ_REQ_PDU from the master to the slave. A changed read request shall differ from the old request in its Invoke ID. Read requests on this connection with the same Invoke ID result in

the reading of the READ_RES_PDU out of the Image Data Memory. A READ_REQ_PDU is not sent in this case.

The slave stores the received read request specific to the connection in the IDM. A changed read request overwrites the old one.

The Layer 2 service CSRD generates a continuous sending of SRD_REQ_PDUs to the slave (polling). The SRD_REQ_PDU of the first or a new read request contains a READ_REQ_PDU. This READ_REQ_PDU is passed to the LLI of the slave in the FDL_DATA_REPLY.ind and is stored in the IDM. Additionally the LLI passes a DTC.ind to FMS. The FMS maps the DTC.ind onto a Read.ind and passes it to the user. When the user of the slave has given the Read.res to FMS, the FMS generates a READ_RES_PDU and passes it to LLI with a DTC.res. The LLI writes the READ_RES_PDU into the low prior Layer 2 update memory (FDL_REPLY_UPDATE.req <Low>). The following SRD_REQ_PDU results in the reading out of the update memory. The SRD_RES_PDU transports the READ_RES_PDU to the master. After reading the update memory (FDL_DATA_REPLY.ind) the LLI passes the stored READ_REQ_PDU with a DTC.ind to FMS. If the user of the slave cannot provide the data until a SRD_REQ_PDU arrives, then the SRD_RES_PDU does not contain a READ_RES_PDU (empty polling).

The LLI of the slave shall activate the RSAP with parameter Indication_mode = "Data" (FMA1/2_RSAP_ACTIVATE.req), so that the FDL_DATA_REPLY.ind is omitted during the empty polling. The LLI of the master shall activate the Poll List LSAP with parameter Confirm_mode = "Data" (FMA1/2_SAP_ACTIVATE.req), so that the FDL_CYC_DATA_REPLY.con is omitted during the empty polling.

The LLI of the master checks the Invoke ID of a received READ_RES_PDU. For a read request with a changed Invoke ID this PDU is ignored, if the Invoke ID is not identical with that of the read request which has been sent. For a read request with an unchanged Invoke ID this leads to a connection release by the LLI. If the Invoke ID is identical the received PDU is stored in the IDM. The first PDU received with the identical Invoke ID is passed to FMS with a DTC.con. The FMS maps the DTC.con onto a Read.con.

The correct execution is controlled by the connection monitoring.

Depending on the actualization of the IDM and the frequency of the Read.req demands in the master different communication characteristics arise:

a) If the data in the IDM is actualized more often than the user of the master reads it, then the data in the IDM, which is not read, will be overwritten.

b) If the data in the IDM is actualized less often than the user of the master reads it, then the user is given the same data repeatedly.

Depending on the frequency of the data recording in the slave the following communication characteristics arise:

a) If the user of the slave makes the data available between two SRD_REQ_PDUs, then the data is transmitted with the SRD_RES_PDU (synchronous update).

If the user of the slave can make the data available repeatedly between two SRD_REQ_PDUs, then the last recorded data is only transmitted in the SRD_RES_PDU if provided for in the realization of Layer 2 and Layer 7. The necessary functions are not described here.

b) If the user of the slave makes no data available between two SRD_REQ_PDUs, then the SRD_RES_PDU is transmitted without any data.

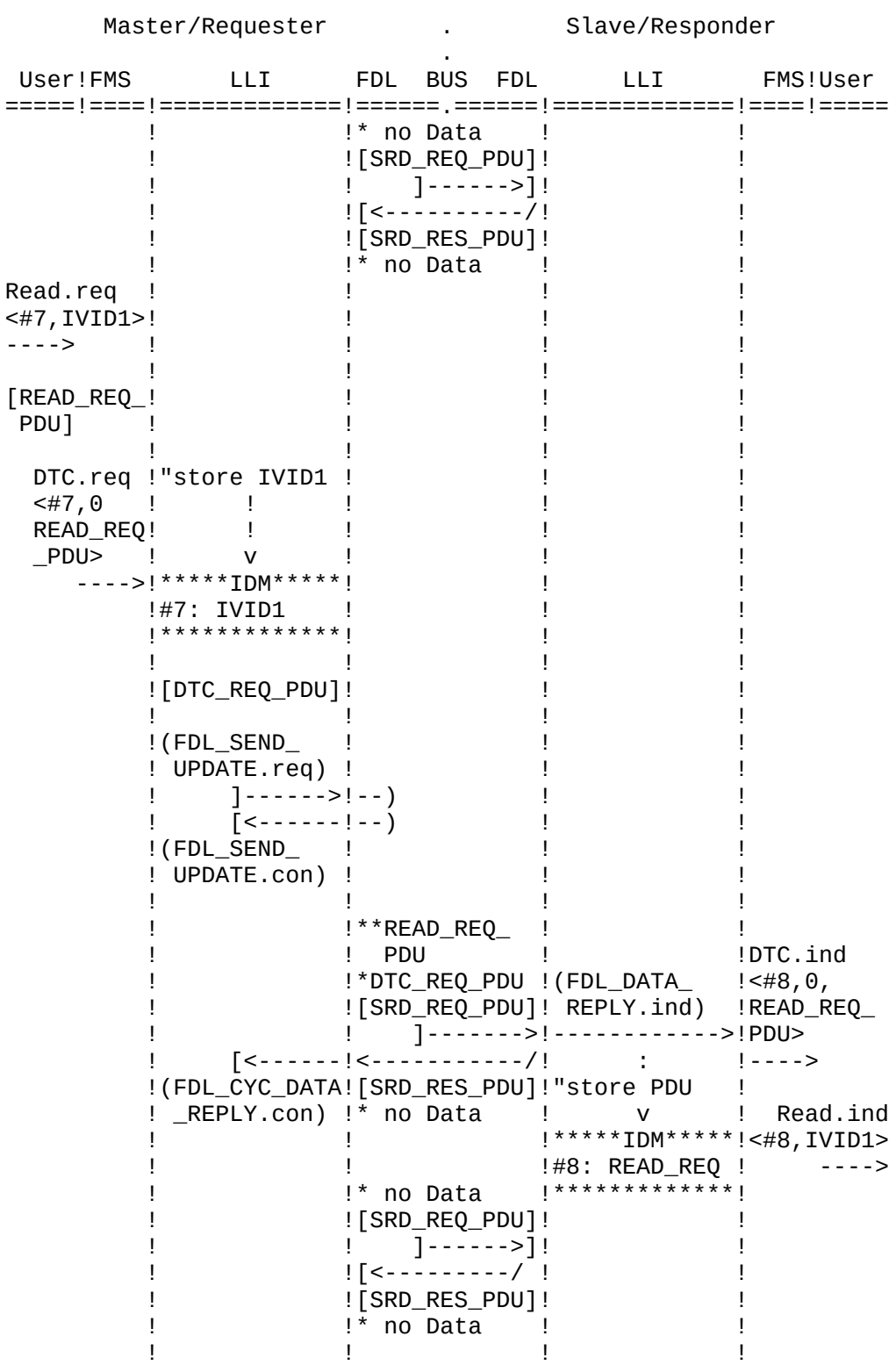


Figure 63. Master-Slave Communication Relationship / Connection for Cyclic Data Transfer with no Slave Initiative / first Read or Read with changed Invoke ID / Request part

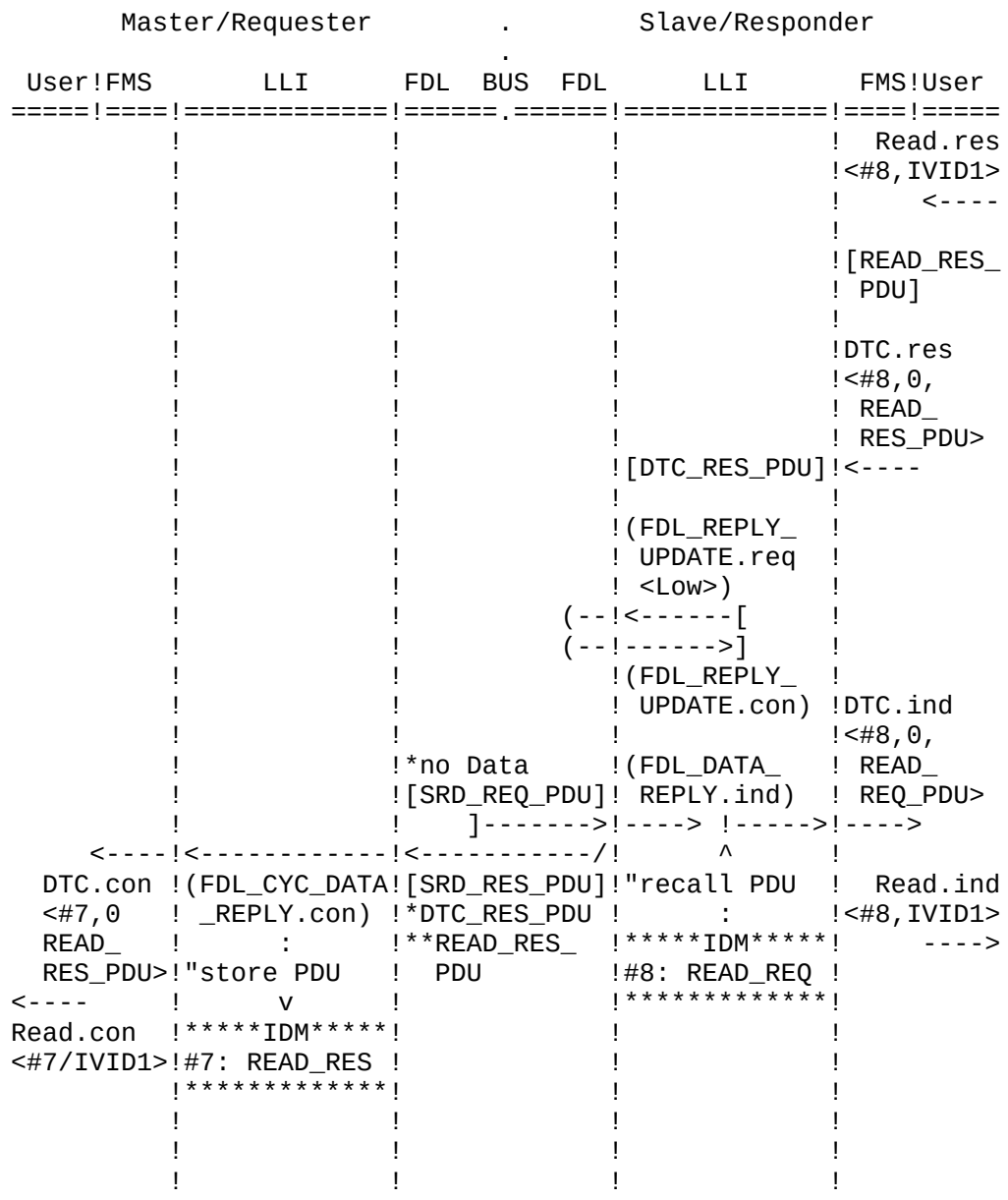


Figure 64. Master-Slave Communication Relationship / Connection for Cyclic Data Transfer with no Slave Initiative / first Read or Read with changed Invoke ID / Response part

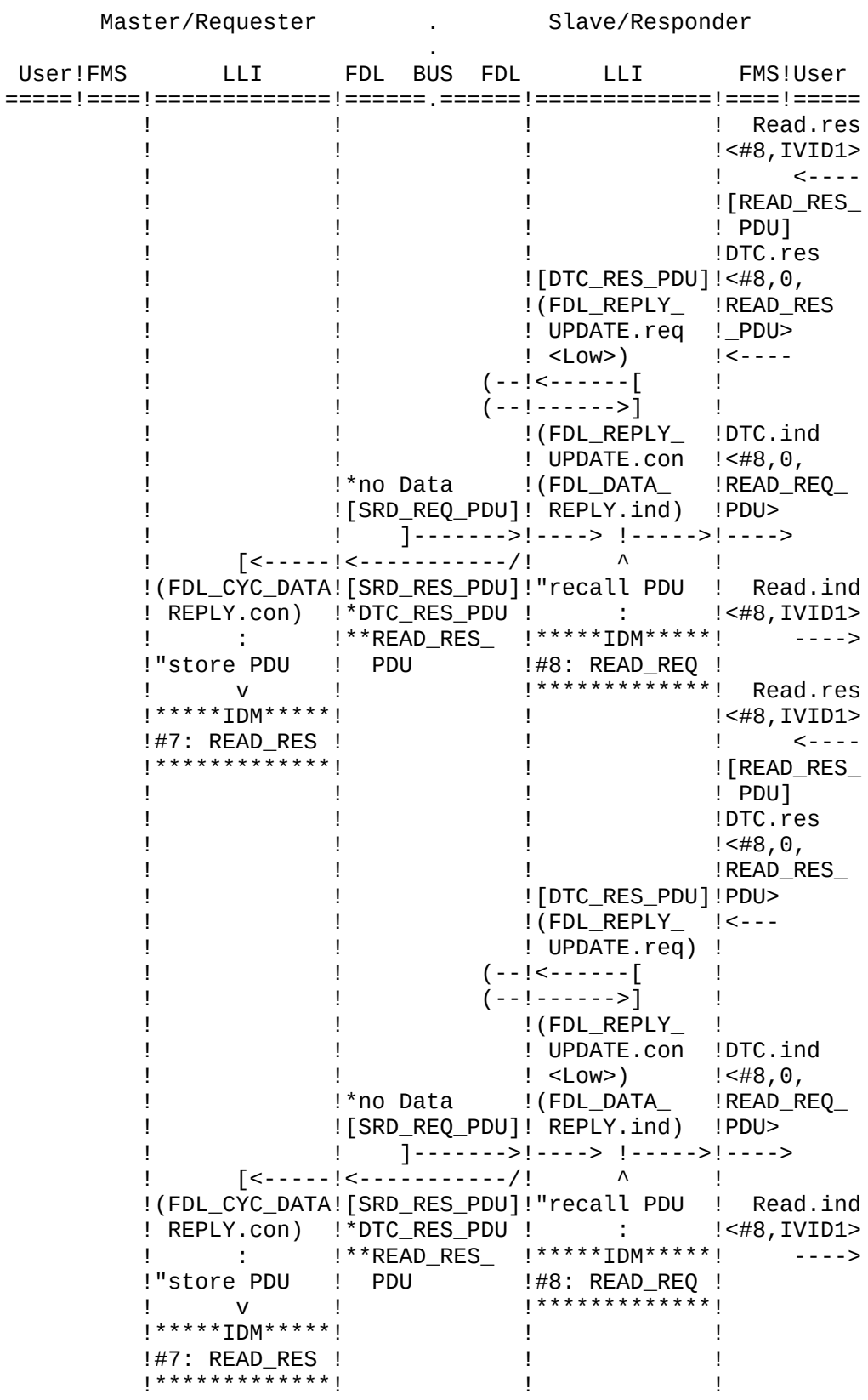


Figure 65. Master-Slave Commun. Relationship / Connection for Cyclic Data Transfer with no Slave Initiative / new Values from Slave - no Request from Master

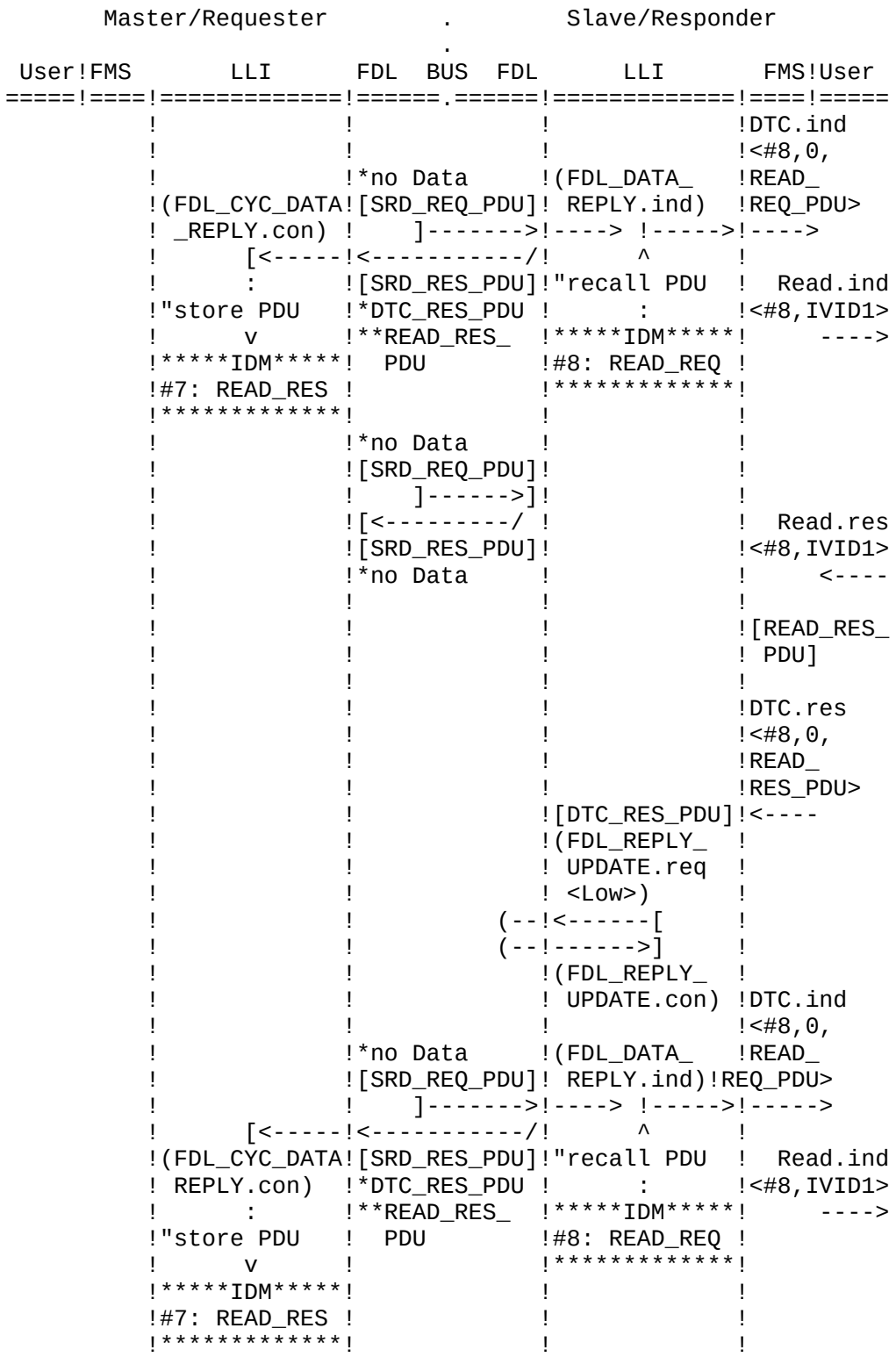


Figure 66. Master-Slave Communication Relationship / Connection for Cyclic Data Transfer with no Slave Initiative / no Values from Slave between SRD_REQ_PDUs

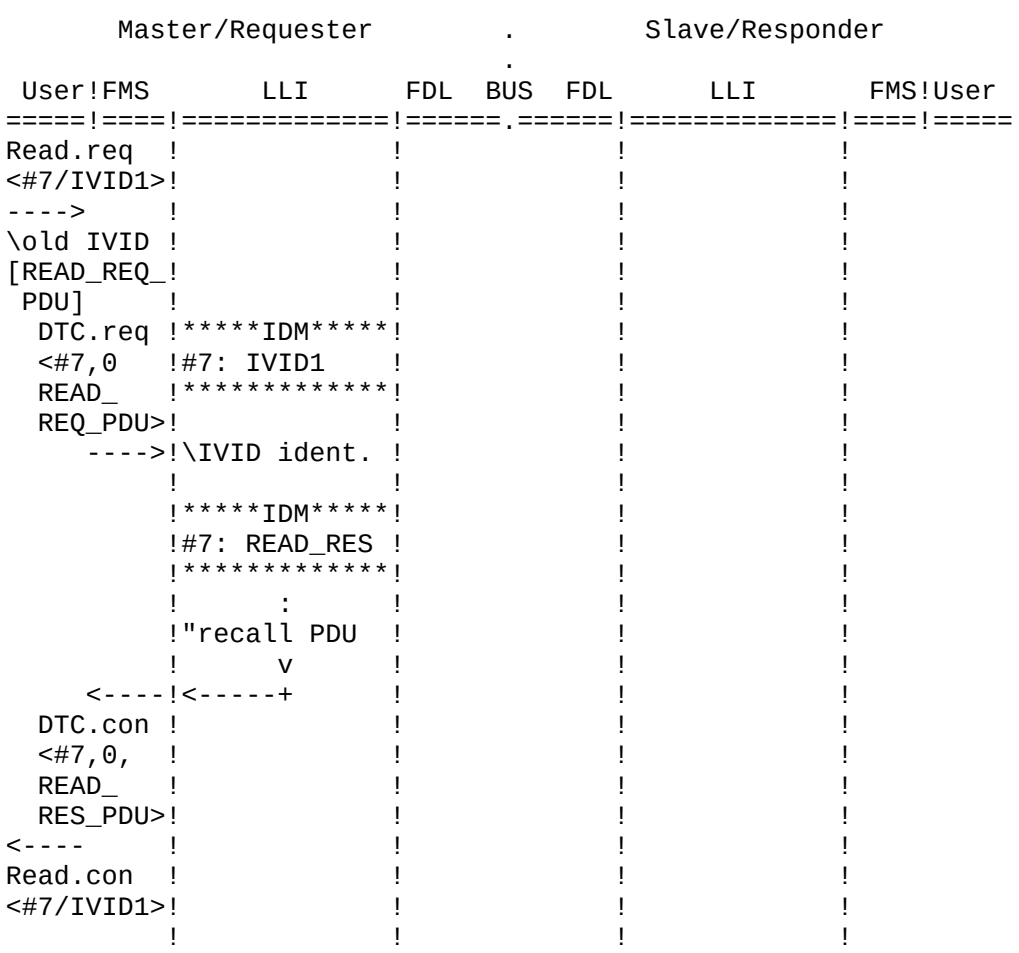


Figure 67. Master-Slave Communication Relationship / Connection for Cyclic Data Transfer with no Slave Initiative / nth Read with the same Invoke ID

Write service:

On a connection for Cyclic Data Transfer with no Slave Initiative the Write service is used for cyclical writing of data into a slave. For each connection only one write request (Write.req) is permissible at a time. The user marks each write request with a special identification (Invoke ID). In this way it is possible to relate the sent write request to the received write confirmation (Write.con). FMS maps a write request (Write.req) from the user onto the LLI service primitive DTC.req. The Invoke ID is stored in the IDM in the LLI of the master. Each write request results in a transmission of the WRITE_REQ_PDU from the master to the slave. A changed write request shall differ from the old request in its Invoke ID. Write requests on this connection with the same Invoke ID cause the write confirmation to be read out of the IDM in the LLI of the master after the next FDL_CYC_DATA_REPLY.con. A WRITE_REQ_PDU is sent in this case.

The received write request (WRITE_REQ_PDU) is stored in the IDM of the slave specific to the connection. Each received WRITE_REQ_PDU overwrites the WRITE_REQ_PDU stored in the IDM of the slave.

The Layer 2 service CSRD in the master causes a continual sending of SRD_REQ_PDUs to the slave (polling). For each write request the FDL_DATA_REPLY.ind contains a WRITE_REQ_PDU. This PDU is stored in the LLI and the FMS receives a DTC.ind. The FMS maps the DTC.ind onto a Write.ind and passes it to the user. When the user of the slave has given the Write.res to the FMS, the FMS generates a WRITE_RES_PDU and passes it to the LLI with a DTC.res. The LLI writes the WRITE_RES_PDU into the low priority Layer 2 update memory (FDL_REPLY_UPDATE.req <Low>). The following SRD_REQ_PDU causes the update memory to be read. The SRD_RES_PDU transports the WRITE_RES_PDU to the master. After reading the update memory (FDL_DATA_REPLY.ind) the LLI passes the stored

WRITE_REQ_PDU with a DTC.ind to the FMS. If the user of the slave cannot provide the WRITE_RES_PDU before a SRD_REQ_PDU arrives, then the SRD_RES_PDU does not contain a WRITE_RES_PDU (empty polling).

The LLI of the master checks the Invoke ID of a received WRITE_RES_PDU. For a write request with a changed Invoke ID this PDU is ignored, if the Invoke ID is not identical with that of the sent write request. For a write request with an unchanged Invoke ID this leads to a connection release by the LLI. If the Invoke ID is identical, the received PDU is stored in the IDM. The first PDU received with the identical Invoke ID is passed to the FMS with a DTC.con. The FMS maps the DTC.con onto a Write.con.

The proper execution is controlled by the connection monitoring.

Depending on the actualization of the IDM and the frequency of the Write.req requests in the master different communication characteristics arise:

a) If the write confirmations in the IDM are more often actualized than the user of the master reads it, then the write confirmations in the IDM, which are not read, will be overwritten.

b) If the write confirmations in the IDM are actualized less often than the user of the master reads it, then the user is given the same write confirmation repeatedly.

c) If the user of the master makes the data available between two SRD_REQ_PDUs, then the data is transmitted within the SRD_REQ_PDU (synchronous update).

If the user of the master can make the data available repeatedly between two SRD_REQ_PDUs, then the last recorded data is transmitted in the SRD_REQ_PDU, only if provided for in the realisation of Layer 2 and Layer 7. The necessary functions are not described here.

d) If the user of the master makes no data available between two SRD_REQ_PDUs, then the SRD_REQ_PDU is transmitted without any data.

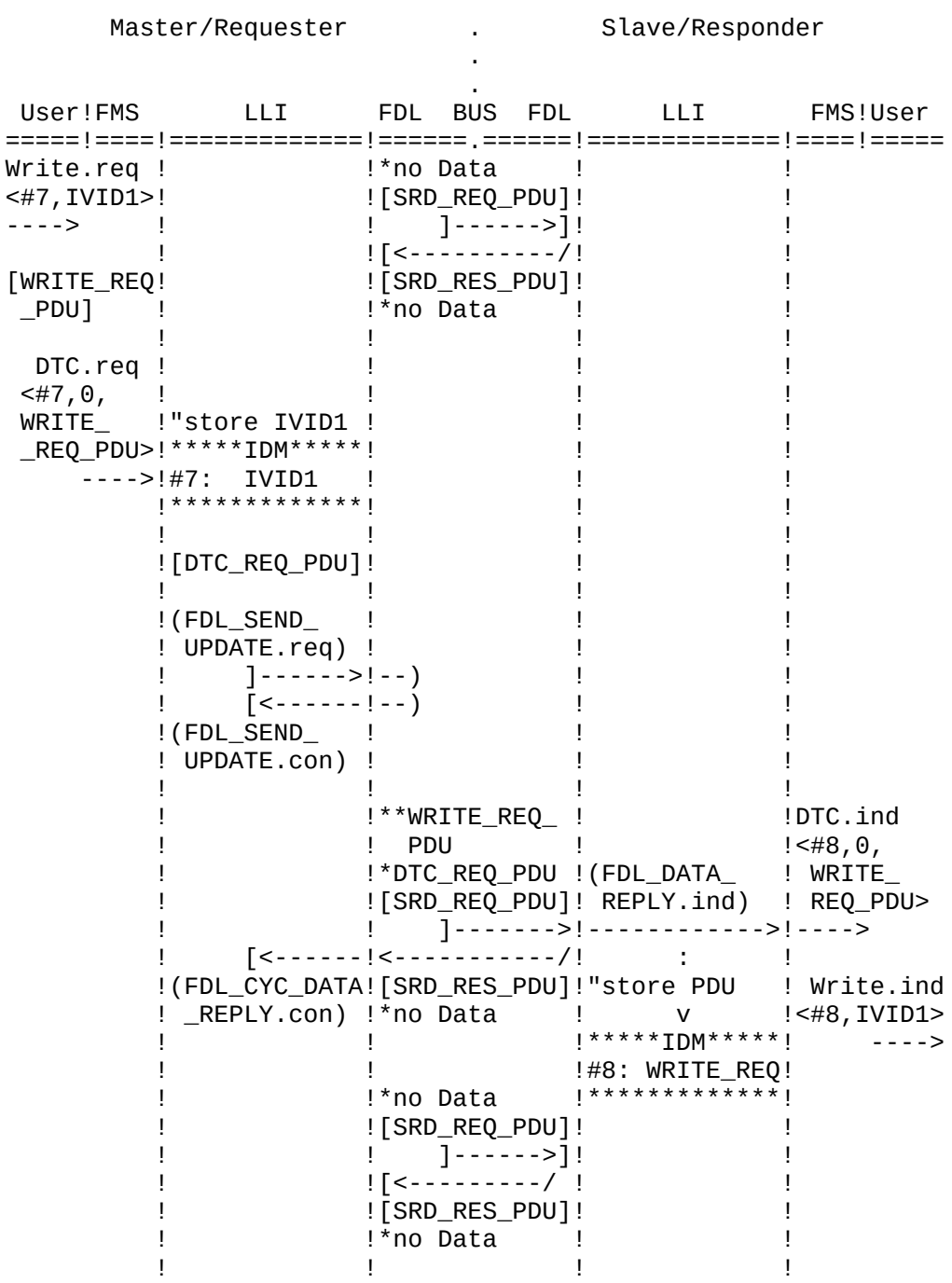


Figure 68. Master-Slave Communication Relationship / Connection for Cyclic Data Transfer with no Slave Initiative / first Write or Write with changed Invoke ID / Request part

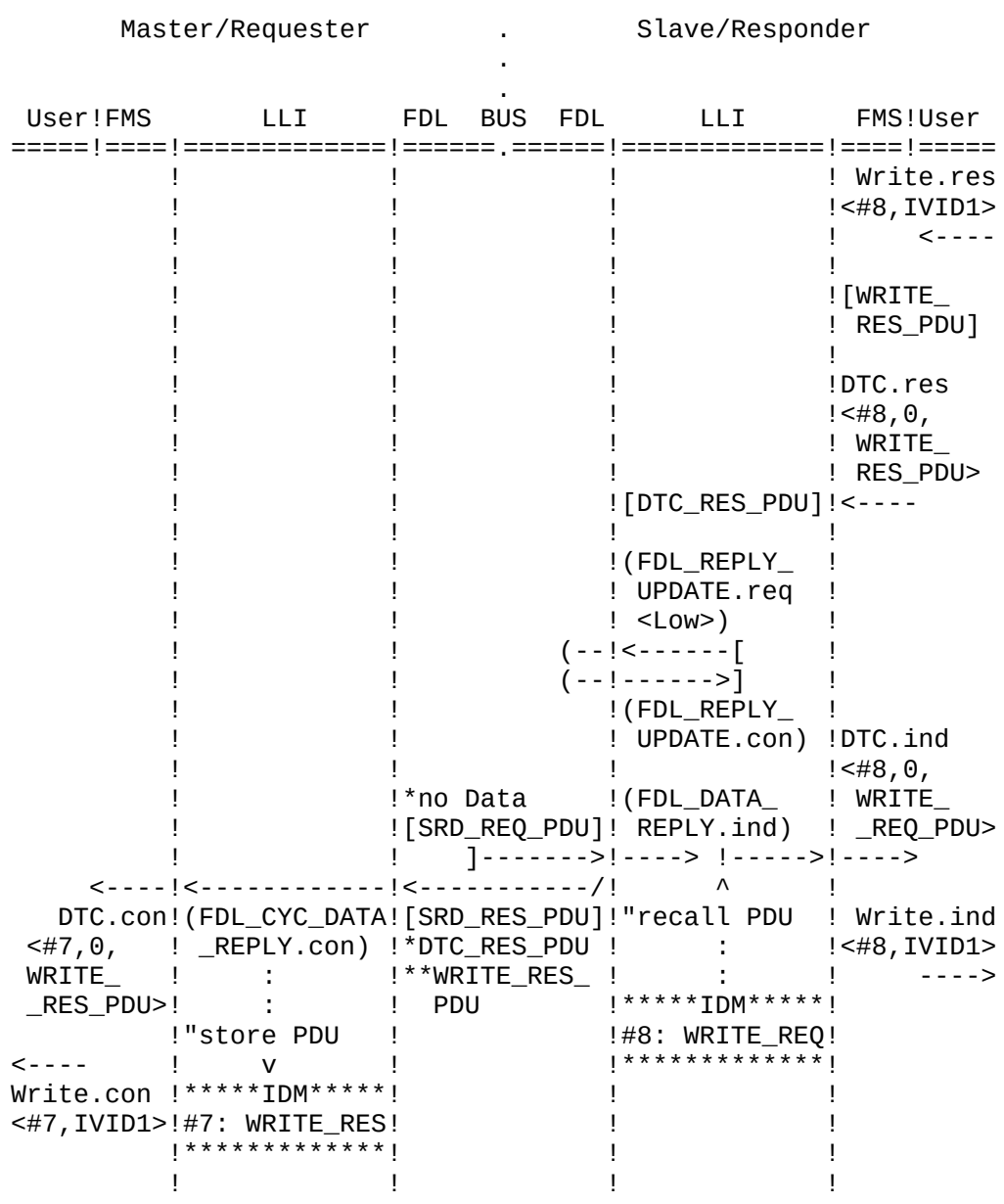


Figure 69. Master-Slave Communication Relationship / Connection for Cyclic Data Transfer with no Slave Initiative / first Write or Write with changed Invoke ID / Response part

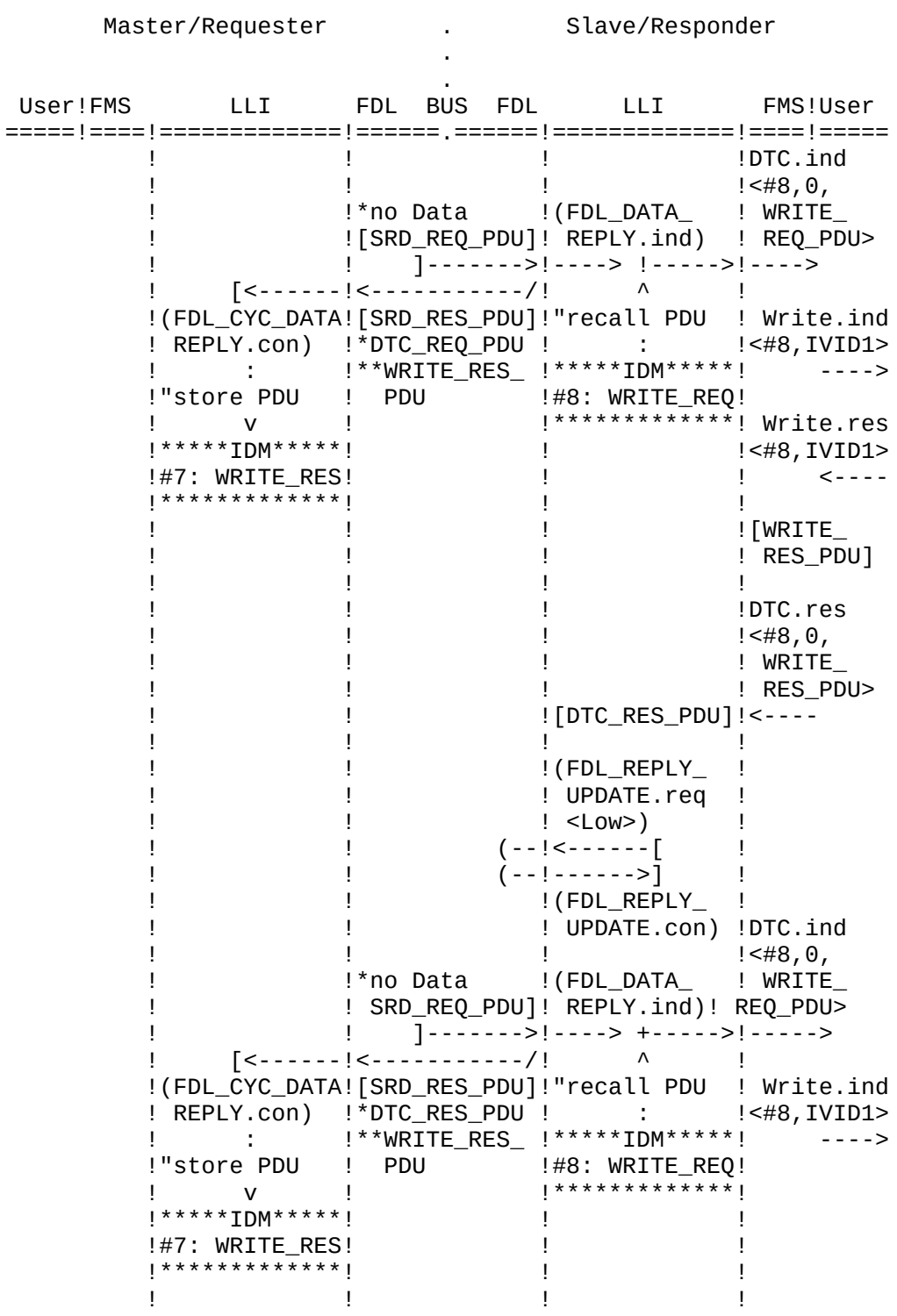


Figure 70. Master-Slave Communication Relationship / Connection for Cyclic Data Transfer with no Slave Initiative / new Write Response from Slave / no Request from Master

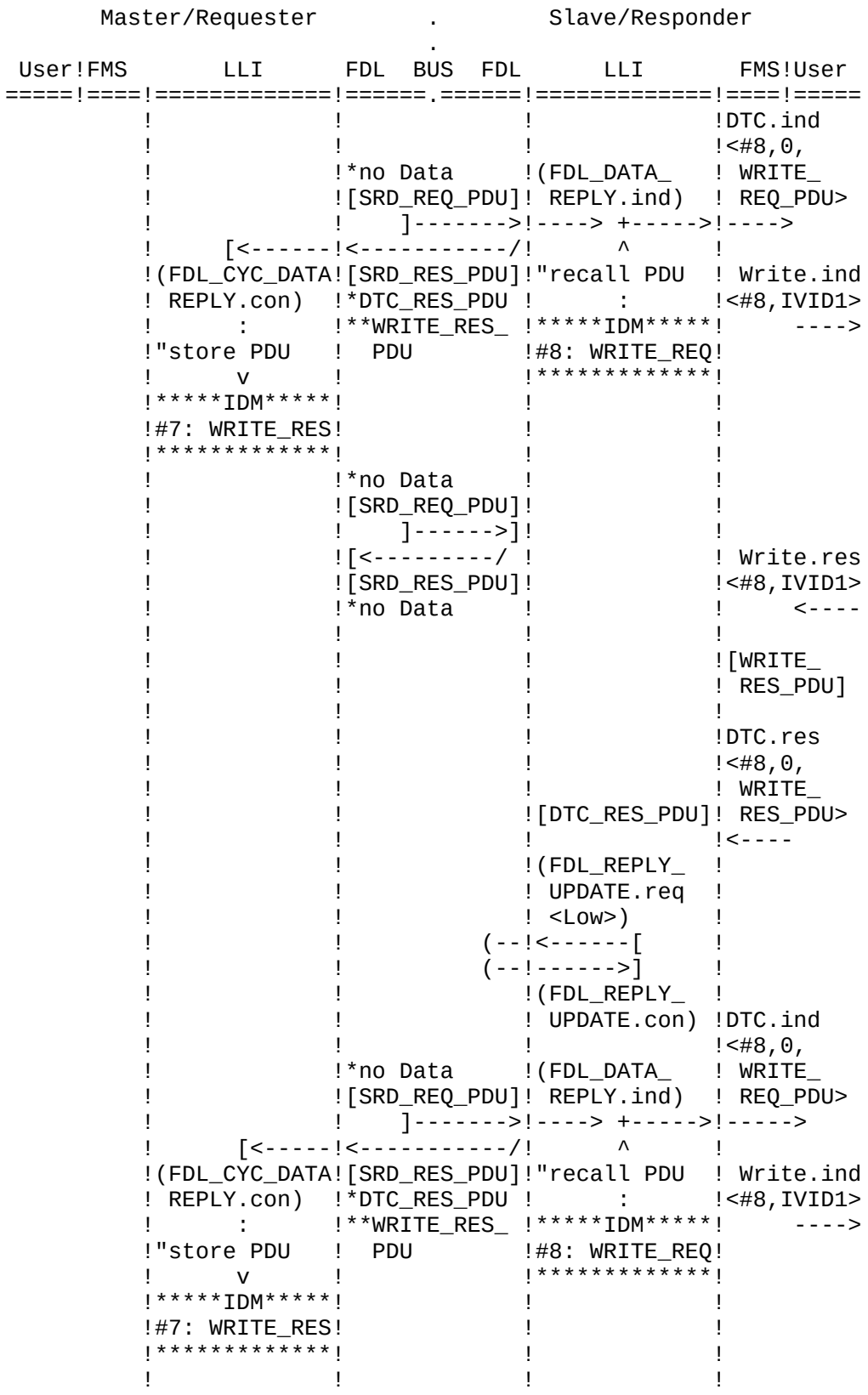


Figure 71. Master-Slave Commun. Relationship / Connection for Cyclic Data Transfer with no Slave Initiative / no Write Response from Slave between SRD_REQ_PDUs

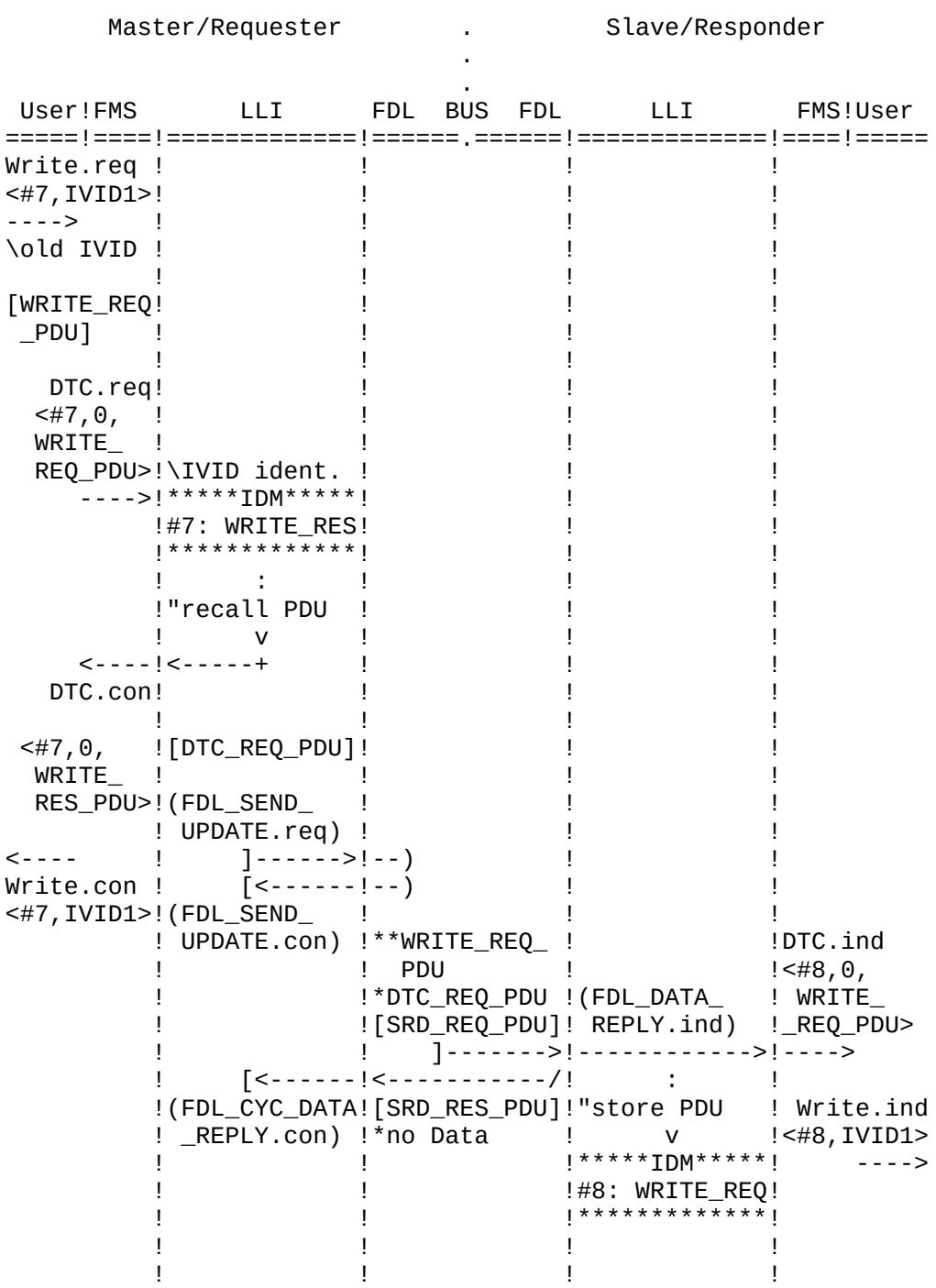


Figure 72. Master-Slave Communication Relationship / Connection for Cyclic Data Transfer with no Slave Initiative / Write with the same Invoke ID / Request part

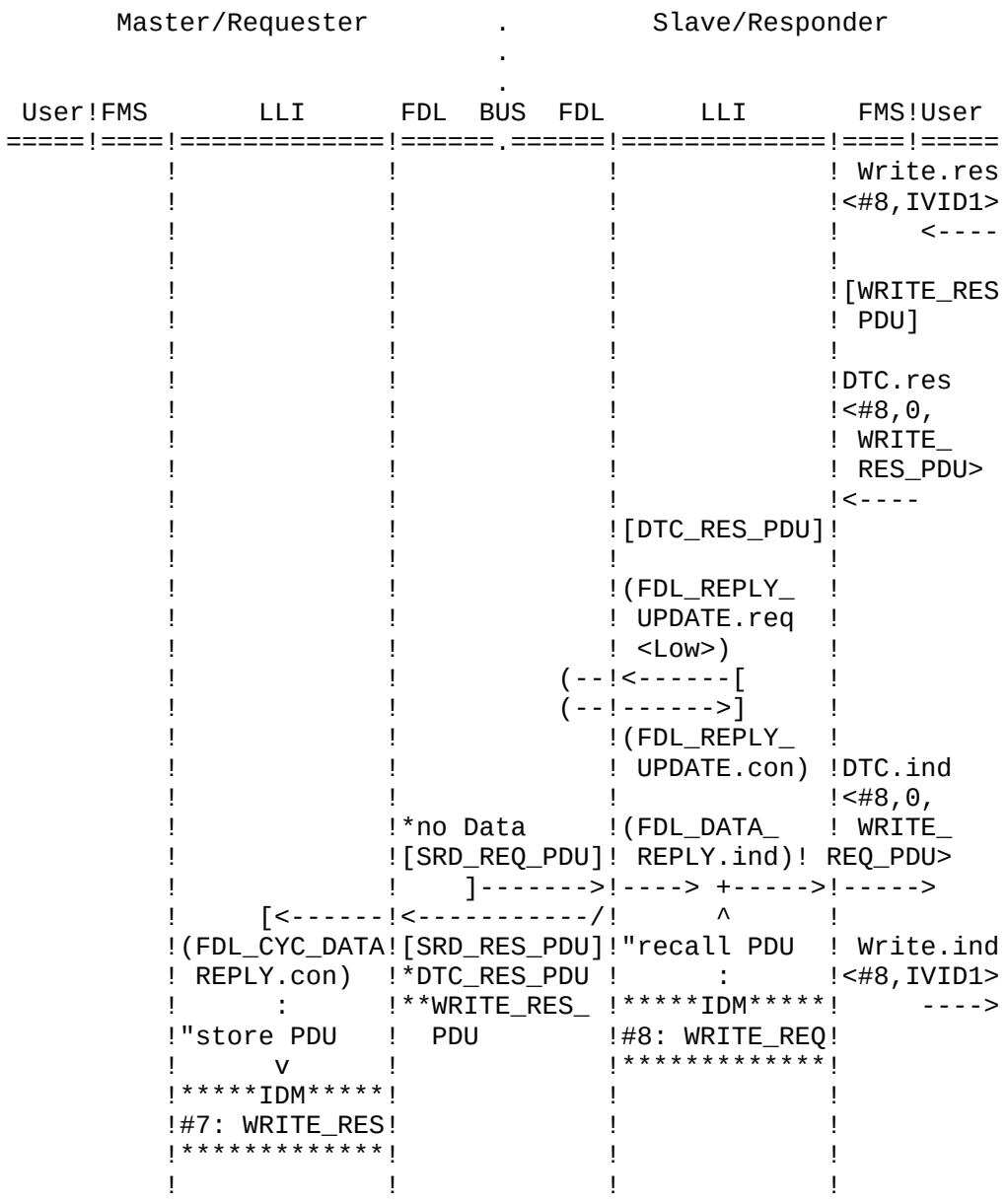


Figure 73. Master-Slave Communication Relationship / Connection for Cyclic Data Transfer with no Slave Initiative / Write with the same Invoke ID / Response part

Unconfirmed FMS services with requester = master:

Unconfirmed FMS services are services which are not confirmed by the user. For these services low or high priority may be used. The mapping of the unconfirmed FMS services onto Layer 2 is described here for the example of the Information-Report service. All other unconfirmed FMS services are mapped in the same way.

InformationReport service:

On a connection for Cyclic Data Transfer with no Slave Initiative the InformationReport service is used for a single writing of data into a slave. FMS maps a request (InformationReport.req) onto the LLI service primitive DTA.req. The priority chosen by the user of the master is transferred transparently. The LLI maps a DTA.req with high priority onto the Layer 2 service SRD with high priority. The LLI maps a DTA.req with low priority onto the Layer 2 service CSRD. Each request causes a transmission of the INFORMATION-REPORT_REQ_PDU with a SRD_REQ_PDU from the master to the slave. The resulting FDL_DATA_REPLY.ind leads to a DTA.ind from the LLI of the slave to FMS. FMS maps this indication onto an InformationReport.ind and passes it to the user.

The SRD_REQ_PDU causes the update memory in the slave to be read. If there is data available (e.g. READ_RES_PDU), this data is transported to the master with the SRD_RES_PDU. If there is no data available in the update memory, then the SRD_RES_PDU contains no data and does not lead to a confirmation to FMS.

If there is memory available again (Buffer_free) for the LLI of the slave to receive another DTA_REQ_PDU, the LLI generates a DTA_ACK_PDU to signal this to the LLI of the master. The LLI stores the DTA_ACK_PDU in the high or low priority Layer 2 update memory (FDL_REPLY_UPDATE.req <Low/High>) corresponding to the priority with which the DTA_REQ_PDU was passed to LLI by the Layer 2. The following SRD_REQ_PDU causes the update memory to be read. The SRD_RES_PDU transports the DTA_ACK_PDU to the LLI of the master. This does not lead to a confirmation to FMS.

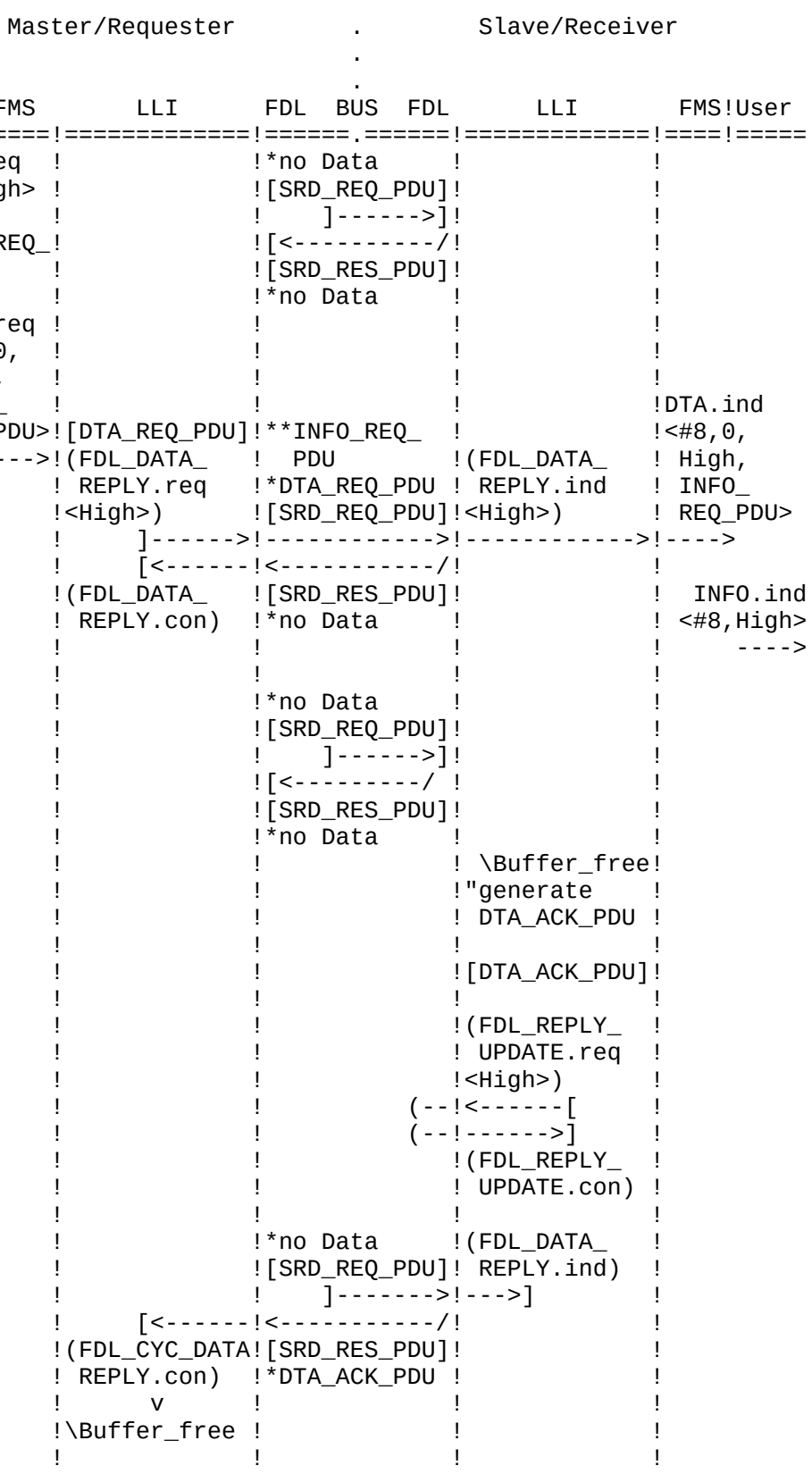


Figure 74. Master-Slave Communication Relationship / Connection for Cyclic Data Transfer with no Slave Initiative / InformationReport with high Priority

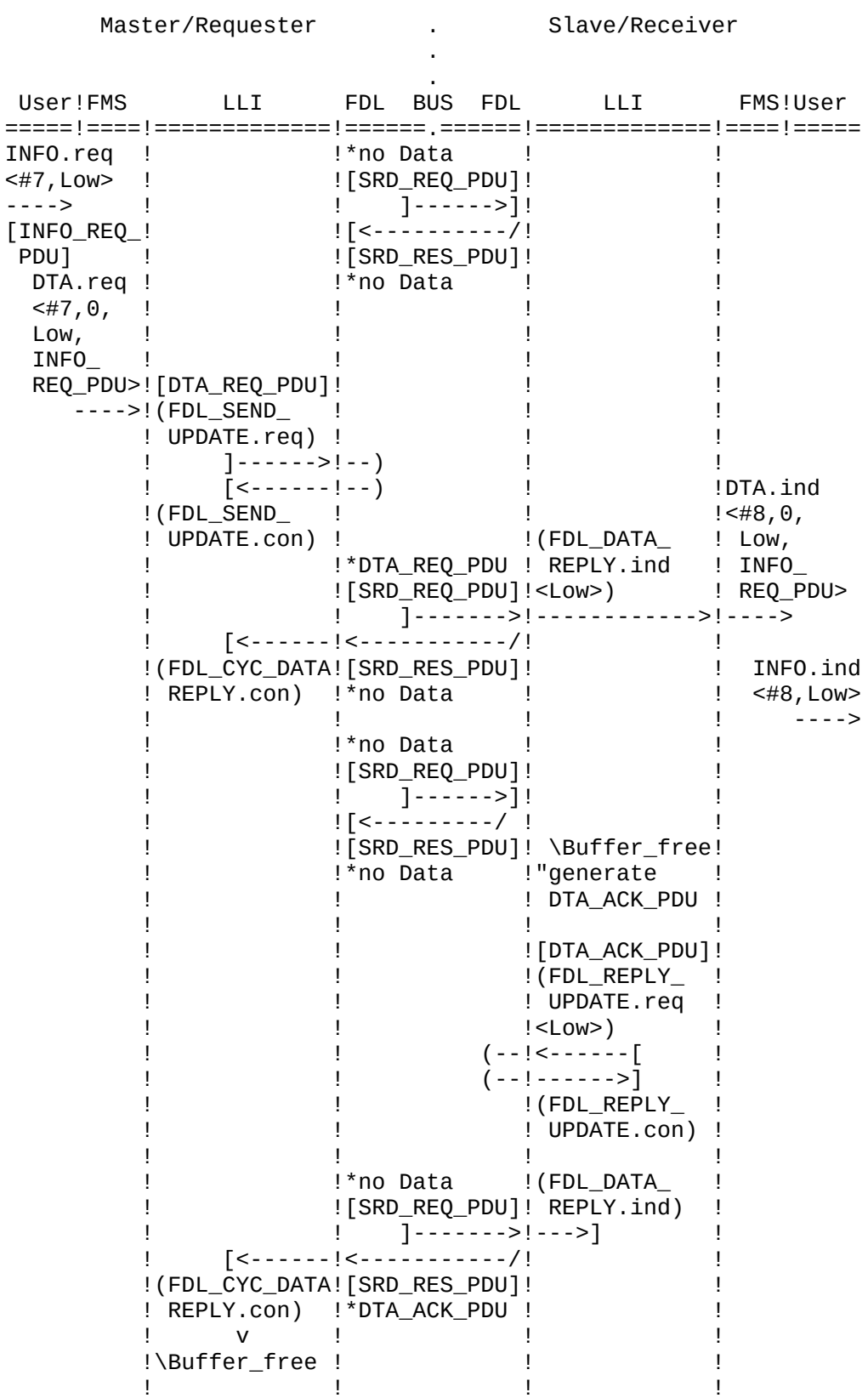


Figure 75. Master-Slave Communication Relationship / Connection for Cyclic Data Transfer with no Slave Initiative / InformationReport with low Priority

Connection for Cyclic Data Transfer with Slave Initiative (MSCY_SI)

On a connection for Cyclic Data Transfer with Slave Initiative the mapping of the confirmed FMS services Read and Write, of the unconfirmed FMS services (requester = master) and of the Reject service is analogous to the mapping for a connection for Cyclic Data Transfer with no Slave Initiative (see subclause 4.3.4.1.1). The transmission of remote FMA7 services is not permitted on connections for Cyclic Data Transfer with no Slave Initiative.

Unconfirmed FMS services with requester = slave:

Unconfirmed FMS services are services which are not acknowledged by the user. For these services low or high priority may be used. The mapping of the unconfirmed FMS services onto Layer 2 is described here for the example of the InformationReport service. All other unconfirmed FMS services are mapped in the same way.

InformationReport service:

On a connection for Cyclic Data Transfer with Slave Initiative the InformationReport service is used for a single writing of data into a master. FMS maps each request (InformationReport.req) onto the LLI service primitive DTA.req. The priority chosen by the user of the slave is transferred transparently. Corresponding to the priority, the LLI stores a DTA_REQ_PDU into the high or low priority Layer 2 update memory. The following SRD_REQ_PDU causes the update memory to be read. The SRD_RES_PDU transports the INFORMATION-REPORT_REQ_PDU to the master. The resulting FDL confirmation leads to a DTA.ind from the LLI of the master to FMS. The FMS maps this DTA.ind onto an InformationReport.ind and passes it to the user.

If there is memory available again (Buffer_free) for the LLI of the master to receive another DTA_REQ_PDU, the LLI generates a DTA_ACK_PDU to signal this to the LLI of the slave. Corresponding to the priority with which the DTA_REQ_PDU was passed to LLI by the Layer 2, LLI maps the DTA_ACK_PDU either onto the Layer 2 service SRD with high priority or onto the Layer 2 service CSRD. The following SRD_REQ_PDU transports the DTA_ACK_PDU to the LLI of the slave. This does not lead to a confirmation to FMS.

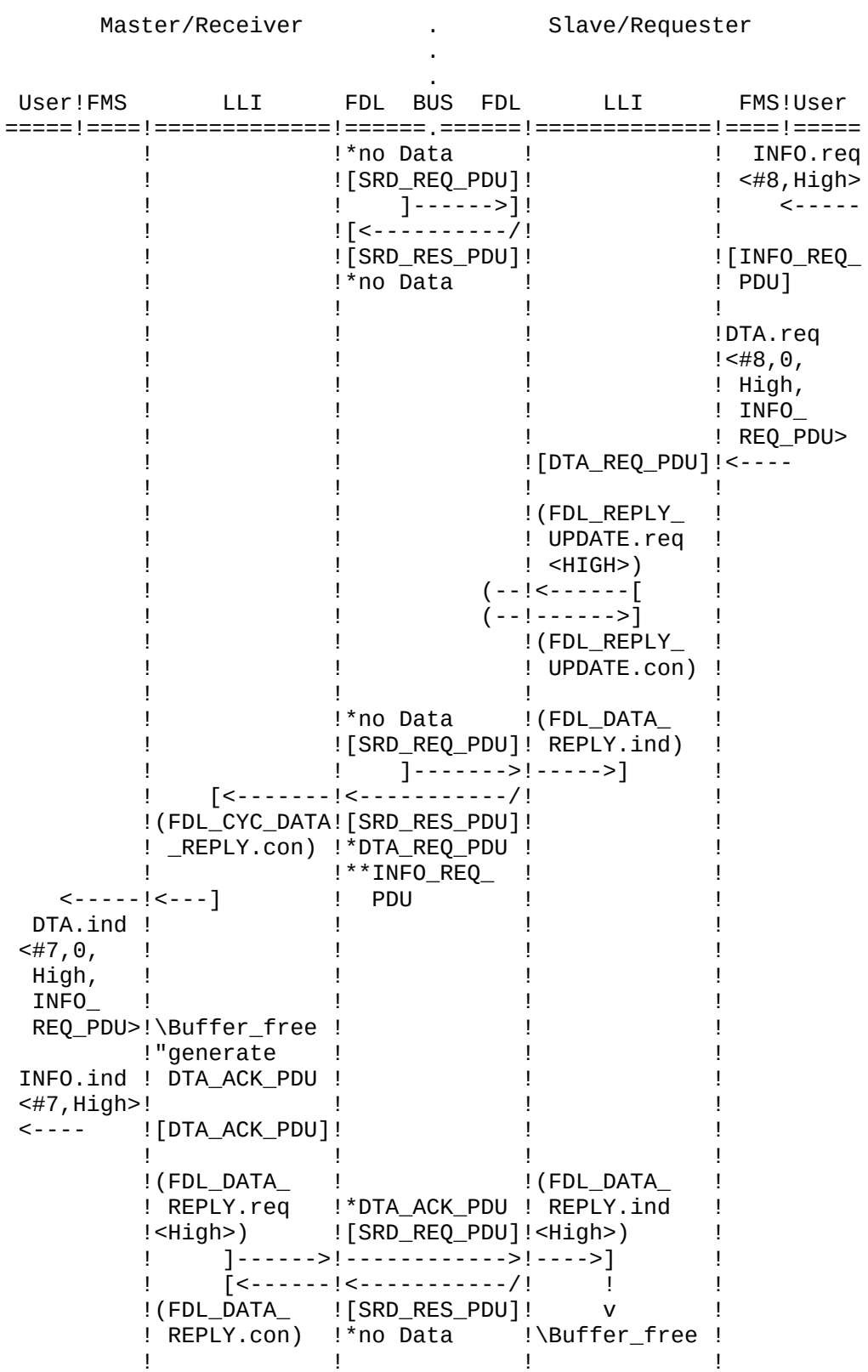


Figure 76. Master-Slave Communication Relationship / Connection for Cyclic Data Transfer with Slave Initiative / InformationReport with high Priority

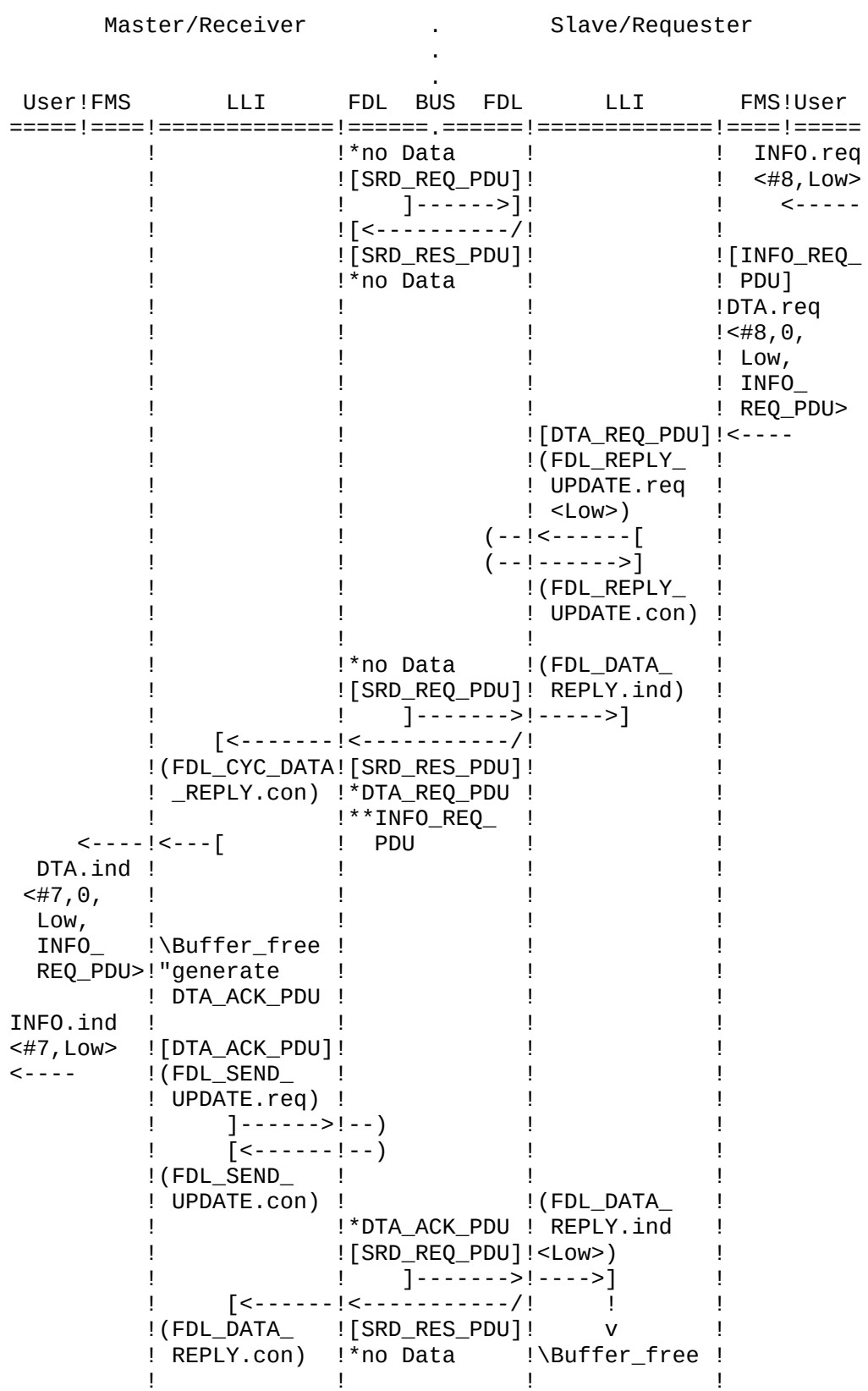


Figure 77. Master-Slave Communication Relationship / Connection for Cyclic Data Transfer with Slave Initiative / InformationReport with low Priority Connection for Acyclic Data Transfer with no Slave Initiative (MSAC)

Connections for Acyclic Data Transfer with no Slave Initiative shall be established like all other connection oriented communication relationships using a connection establishment service (Initiate).

During the subsequent data transfer phase all FMS and all remote FMA7 services, for which the master is requester, are permitted on this connection. Multiple connections for Acyclic Data Transfer with no Slave Initiative may exist to a slave at the same time. All these connections shall use the Poll List LSAP in the master and different LSAPs in the slave.

With the first FMS service (e.g. InformationReport.req) or the first FMA7 service (e.g. FMA7-Read-Value.req) the LLI starts the polling of the slave (FDL_CYC_POLL_ENTRY.req) by the Layer 2 service CSRD on the respective connection. The Layer 2 service CSRD causes a continual sending of SRD_REQ_PDUs to the slave (polling). If the master has received the last outstanding LLI PDU, the polling to the slave is stopped on this connection (FDL_CYC_POLL_ENTRY.req). A new service request from FMS or FMA7 causes LLI to start the polling of the slave again until all outstanding LLI PDUs have been transmitted from the slave to the master.

Optionally the connection may be monitored in the master or the slave.

Connections for Acyclic Data Transfer with no Slave Initiative may be released with the Abort service.

Confirmed FMS services:

Confirmed FMS services are services which are acknowledged by the user. The mapping of the confirmed FMS services onto Layer 2 is described here for the example of the Read service. All other confirmed FMS services are mapped in the same way.

Read service:

On a connection for Acyclic Data Transfer with no Slave Initiative the Read service is used for reading of data from a slave. The user marks each read request (Read.req) with a special identification (Invoke ID). In this way it is possible to relate the sent read request to the received read response. The FMS maps each read request (Read.req) onto the LLI service primitive DTC.req. The READ_REQ_PDU is transferred from the master to the slave with a SRD_REQ_PDU. The resulting FDL_DATA_REPLY.ind leads to a DTC.ind from the LLI of the slave to FMS. The FMS maps the DTC.ind onto a Read.ind to the user.

The user of the slave passes the Read.res to FMS. The FMS generates the READ_RES_PDU and passes it with a DTC.res to LLI. The LLI stores the READ_RES_PDU in the low priority Layer 2 update memory (FDL_REPLY_UPDATE.req <Low>). The following SRD_REQ_PDU causes the update memory to be read. The SRD_RES_PDU transports the data to the master. This leads to a DTC.con from LLI to FMS in the master and then to a Read.con from FMS to the user.

If the user of the slave did not provide the data before a SRD_REQ_PDU arrives, then the SRD_RES_PDU does not contain any data and does not lead to a DTC.con (empty polling).

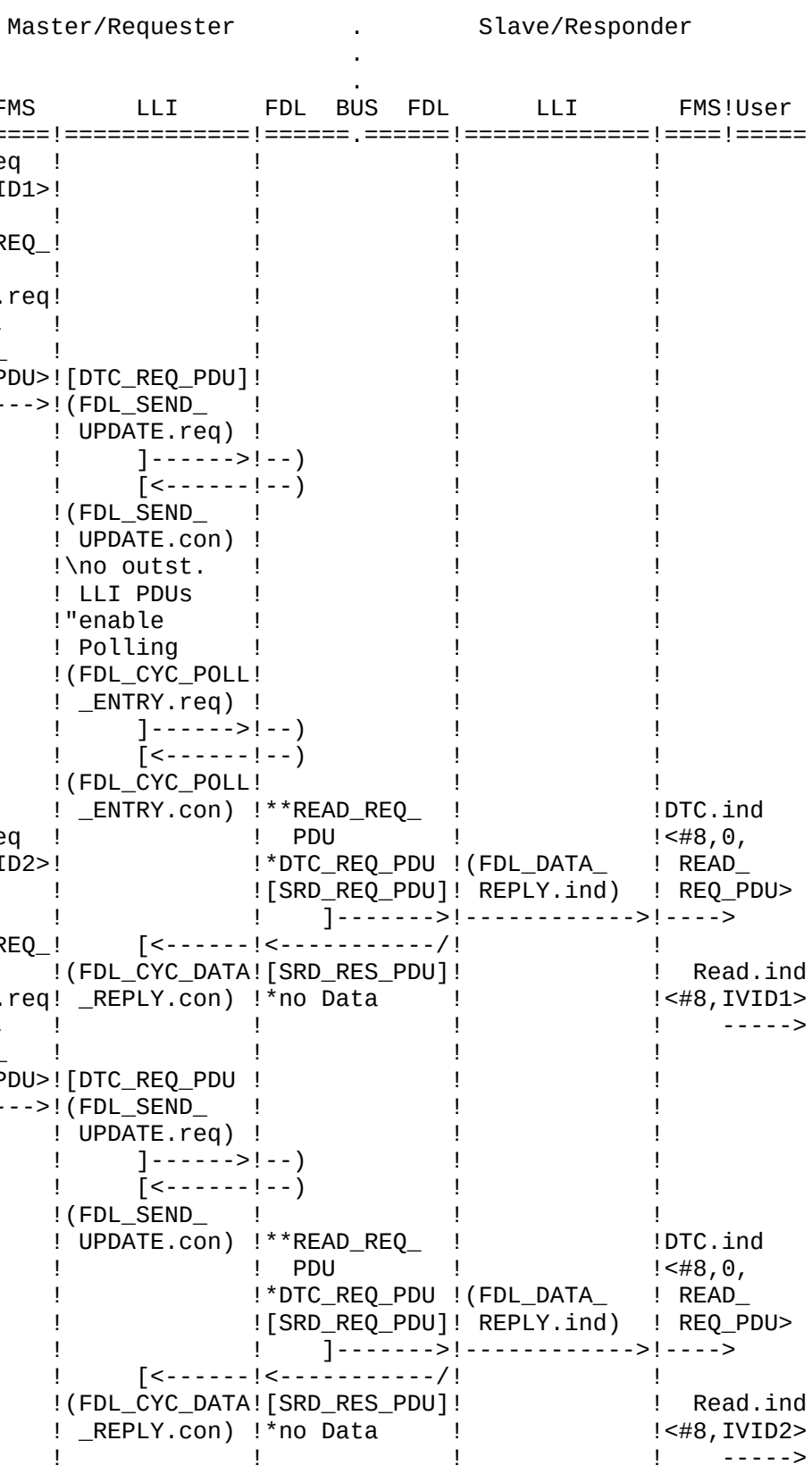


Figure 78. Master-Slave Communication Relationship / Connection for Acyclic Data Transfer with no Slave Initiative / parallel Read Services / Request part

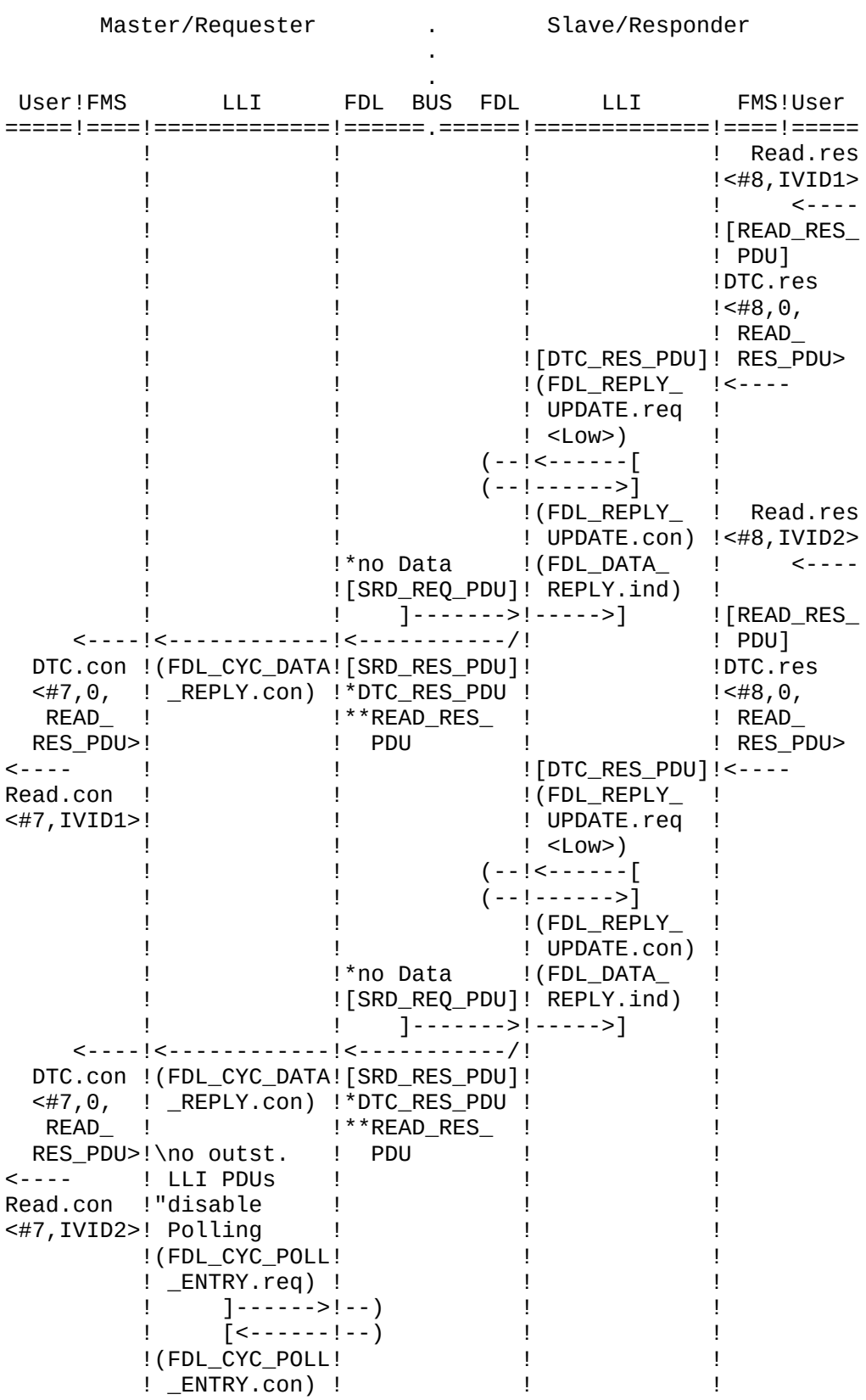


Figure 79. Master-Slave Communication Relationship / Connection for Acyclic Data Transfer with no Slave Initiative / parallel Read Services / Response part

Unconfirmed FMS services with requester = master:

Unconfirmed FMS services are services which are not acknowledged by the user. For these services low or high priority may be used. The mapping of the unconfirmed FMS services onto Layer 2 is described here for the example of the InformationReport service. All other unconfirmed FMS services are mapped in the same way.

InformationReport service:

On a connection for Acyclic Data Transfer with no Slave Initiative the InformationReport service is used for a single writing of data into a slave. The FMS maps each request (InformationReport.req) onto the LLI service primitive DTA.req. The priority chosen by the user of the master is transferred transparently. The LLI maps a DTA.req with high priority onto the Layer 2 service SRD with high priority. The LLI maps a DTA.req with low priority onto the Layer 2 service CSRD. Each request causes a transmission of the INFORMATION-REPORT_REQ_PDU with a SRD_REQ_PDU from the master to the slave. The resulting FDL_DATA_REPLY.ind leads to a DTA.ind from the LLI of the slave to FMS. The FMS maps this indication onto an InformationReport.ind and passes it to the user.

The SRD_REQ_PDU causes the update memory in the slave to be read. If there is data available (e.g. READ_RES_PDU), this data is transported to the master with the SRD_RES_PDU. If there is no data available in the update memory, then the SRD_RES_PDU contains no data and does not lead to a confirmation to FMS.

If there is memory available again (Buffer_free) for the LLI of the slave to receive another DTA_REQ_PDU, the LLI generates a DTA_ACK_PDU to signal this to the LLI of the master. The LLI stores the DTA_ACK_PDU into the high or low priority Layer 2 update memory (FDL_REPLY_UPDATE.req <Low/High>) corresponding to the priority with which the DTA_REQ_PDU has been passed to LLI by the Layer 2. The following SRD_REQ_PDU causes the update memory to be read. The SRD_RES_PDU transports the DTA_ACK_PDU to the LLI of the master. This does not lead to a confirmation to FMS.

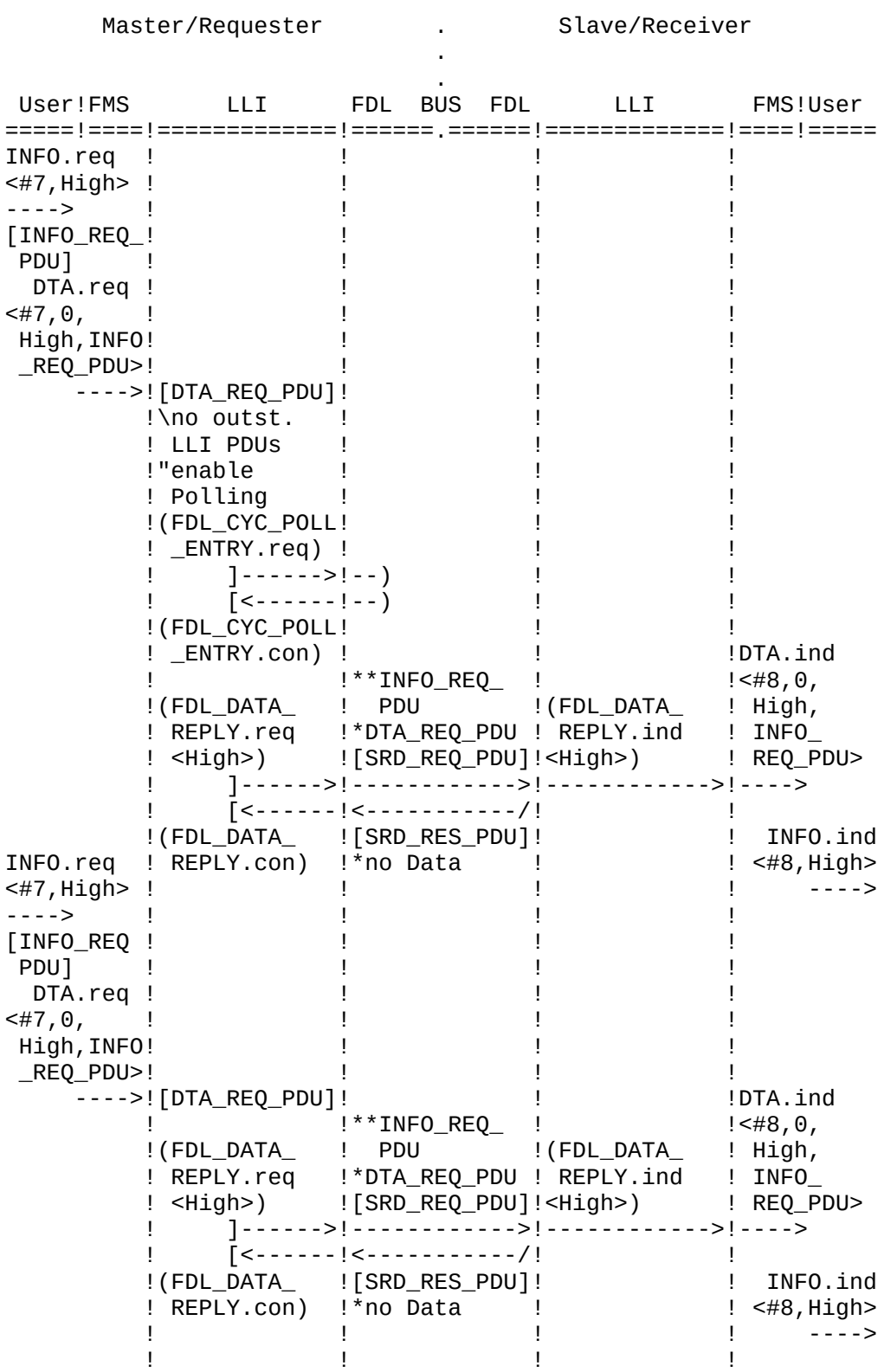


Figure 80. Master-Slave Communication Relationship / Connection for Acyclic Data Transfer with no Slave Initiative / parallel InformationReport Services with high Priority / Request part

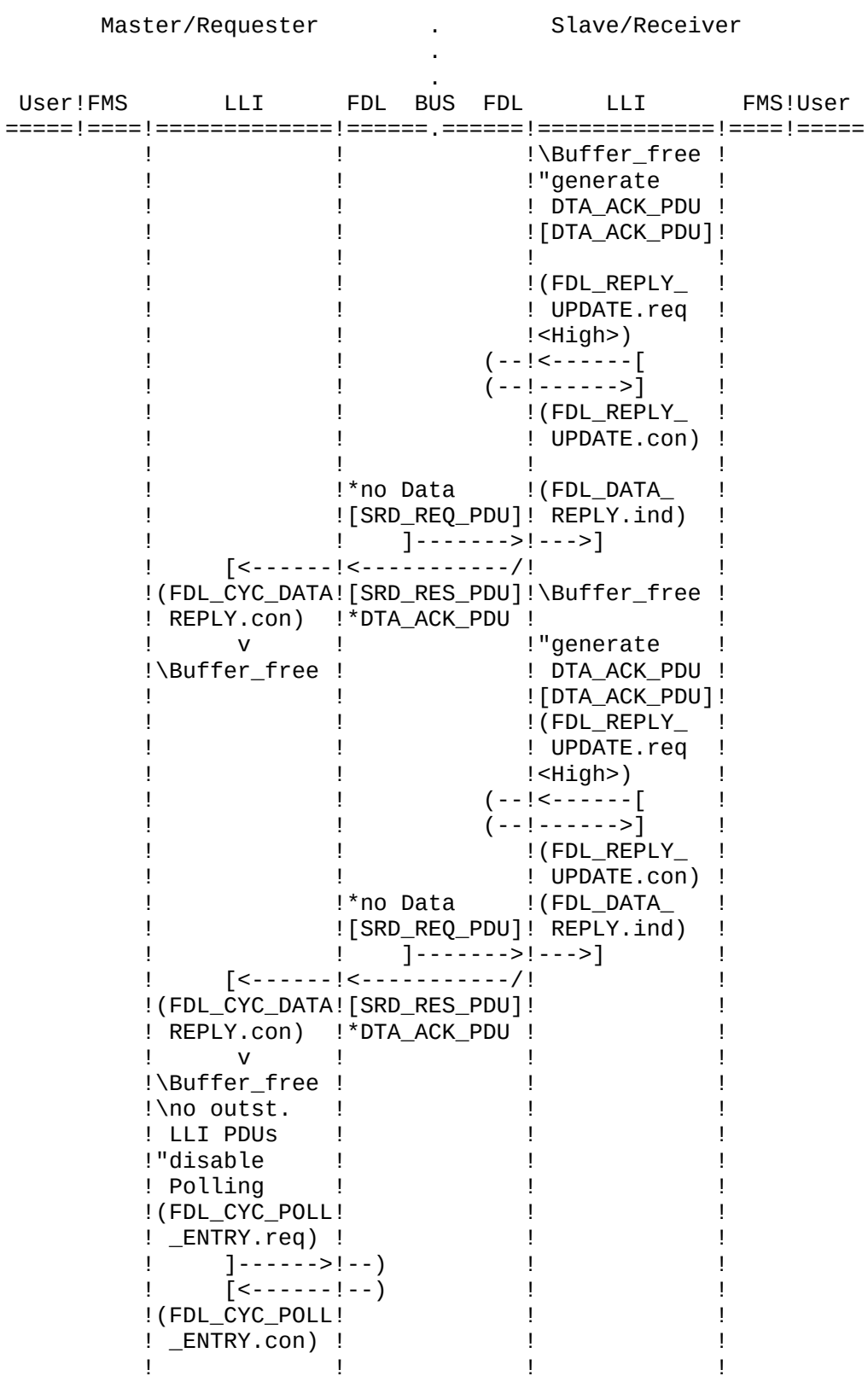


Figure 81. Master-Slave Communication Relationship / Connection for Acyclic Data Transfer with no Slave Initiative / parallel InformationReport Services with high Priority / Receiver part

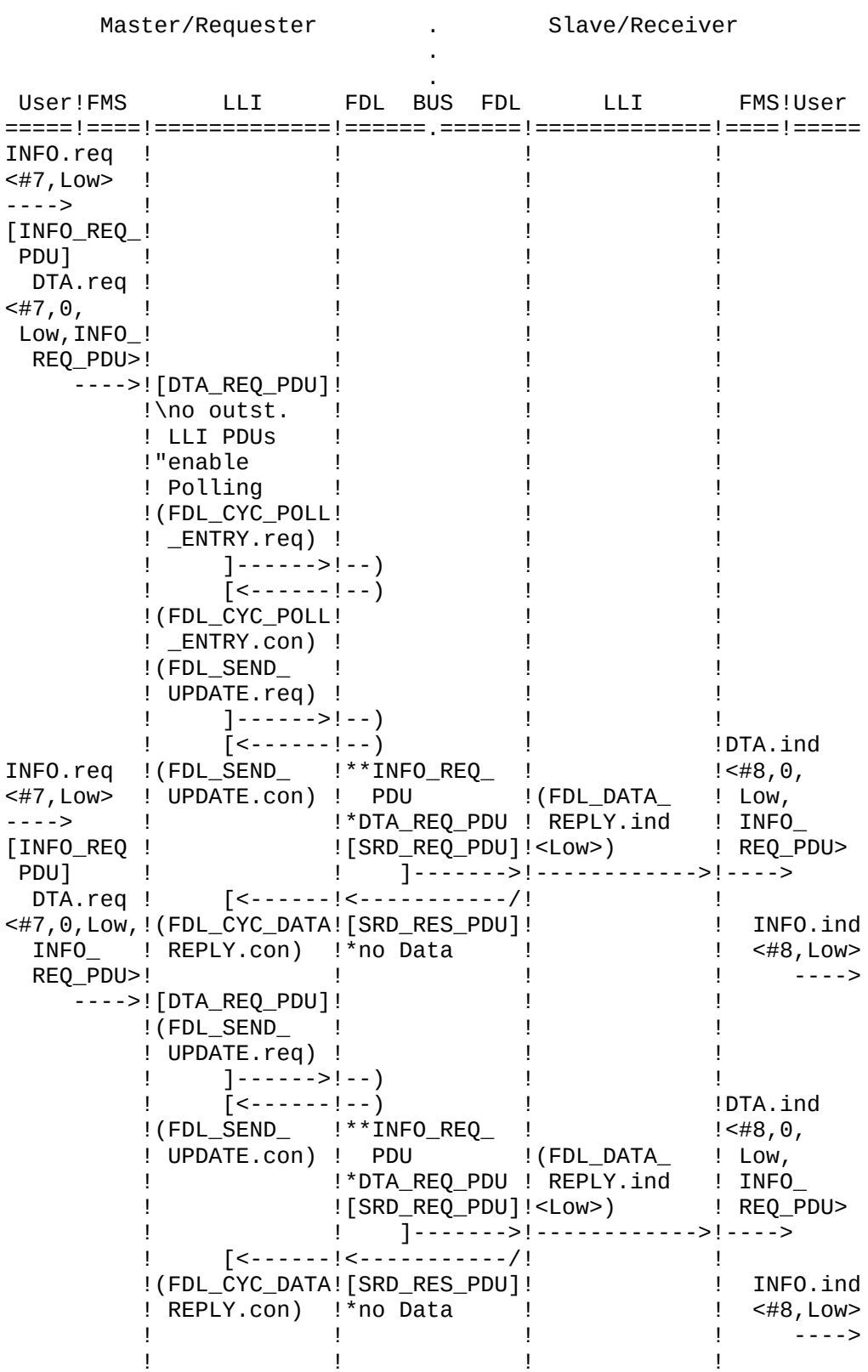


Figure 82. Master-Slave Communication Relationship / Connection for Acyclic Data Transfer with no Slave Initiative / parallel InformationReport Services with low Priority / Request part

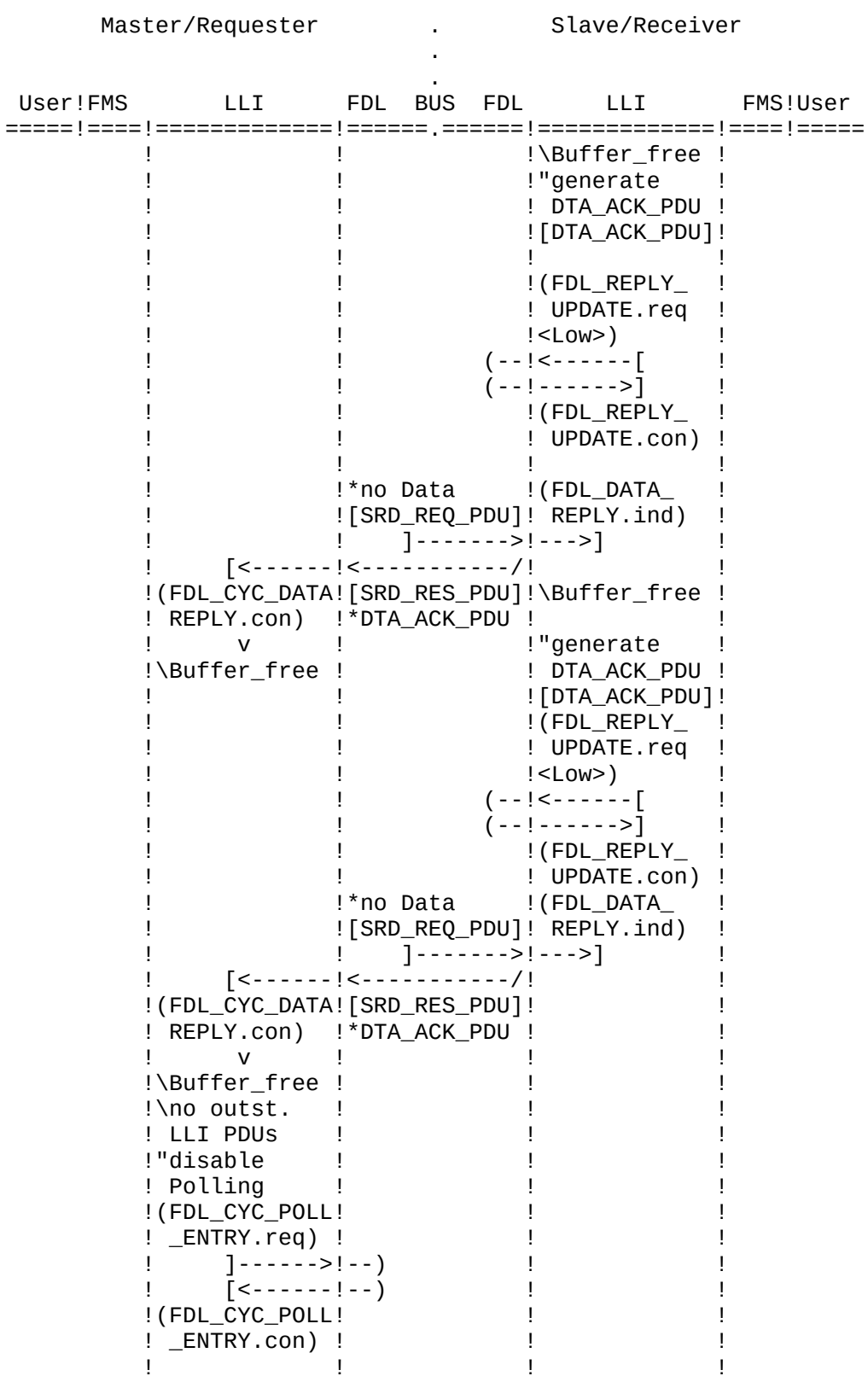


Figure 83. Master-Slave Communication Relationship / Connection for Acyclic Data Transfer with no Slave Initiative / parallel InformationReport Services with low Priority / Receiver part

Remote FMA7 services:

Remote FMA7 services are mapped onto the LLI in the same way as the confirmed FMS services. All FMA7 services are acknowledged by the user. The mapping of the remote FMA7 services onto Layer 2 is described here for the example of the Read Value Rem service. All other remote FMA7 services are mapped in the same way.

Read Value Rem service:

On a connection for Acyclic Data Transfer with no Slave Initiative the Read Value Rem service is used for reading of variables or counters in a slave. FMA7 maps each read request (ReadValueRem.req) onto the LLI service primitive DTC.req. The READ_VALUE-REM_REQ_PDU is transported with a SRD_REQ_PDU from the master to the slave. The resulting FDL_DATA_REPLY.ind leads to a DTC.ind from the LLI of the slave to FMA7. FMA7 maps the DTC.ind onto a ReadValueRem.ind to the user.

The user of the slave passes the ReadValueRem.res to FMA7. FMA7 generates the READ-VALUE-REM_RES_PDU and passes it with a DTC.res to LLI. LLI stores the READ-VALUE-REM_RES_PDU in the low priority Layer 2 update memory (FDL_REPLY_UPDATE.req <Low>). The following SRD_REQ_PDU causes the update memory to be read.

The SRD_RES_PDU transports the READ-VALUE-REM_RES_PDU to the master. This leads in the master to a DTC.con from LLI to FMA7 and then a ReadValueRem.con from FMA7 to the user. If the user of the slave did not provide the data before a SRD_REQ_PDU arrives, then the SRD_RES_PDU does not contain a READ-VALUE-REM_RES_PDU and does not lead to a DTC.con to FMA7 (empty polling).

The following abbreviations are used in the figures below:

RV.xxx : ReadValueRem.xxx service primitive

RV_XXX_PDU : READ-VALUE-REM_XXX_PDU

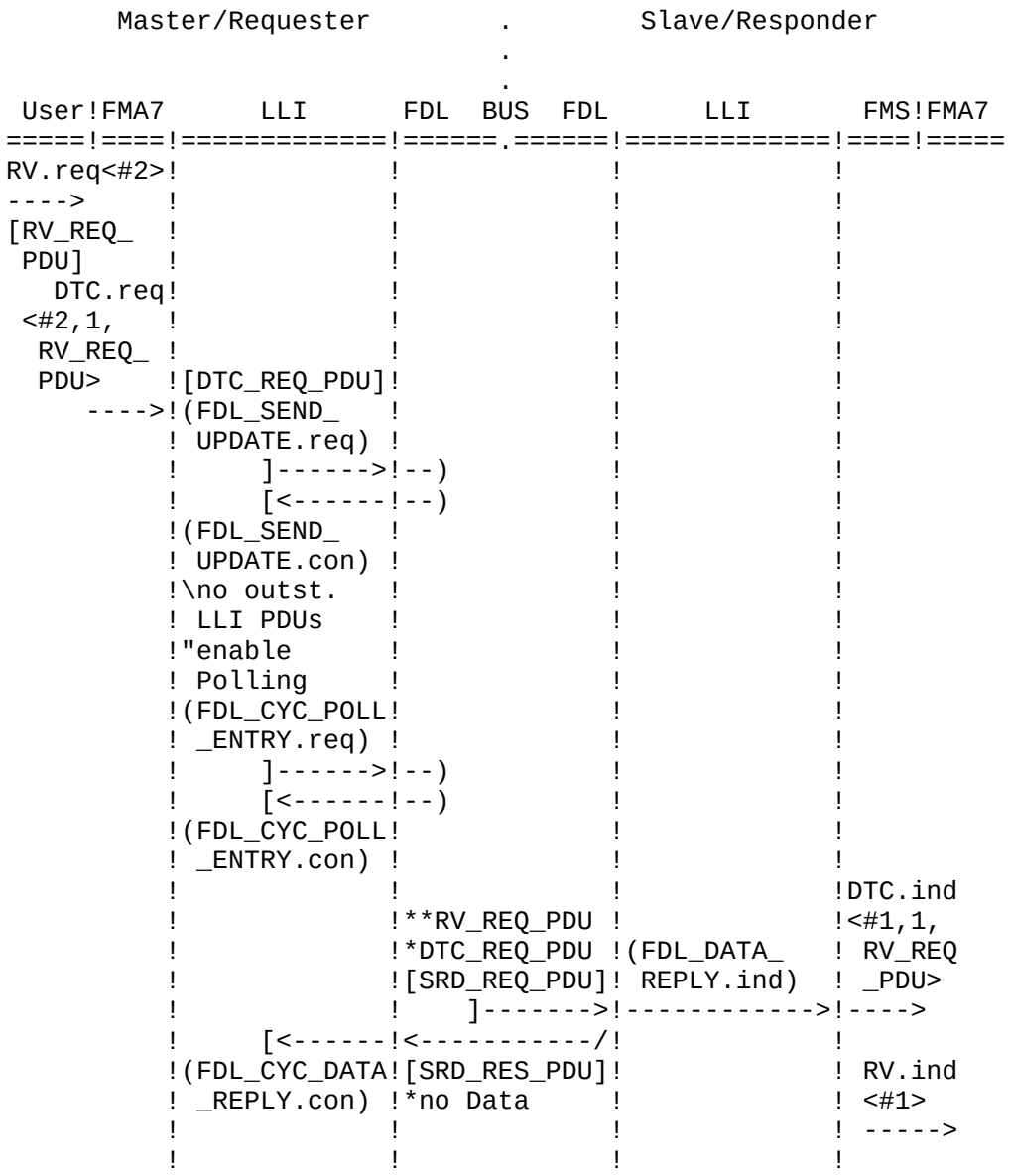


Figure 84. Master-Slave Communication Relationship / Connection for Acyclic Data Transfer with no Slave Initiative / Mapping of the Read Value Rem Service / Request part

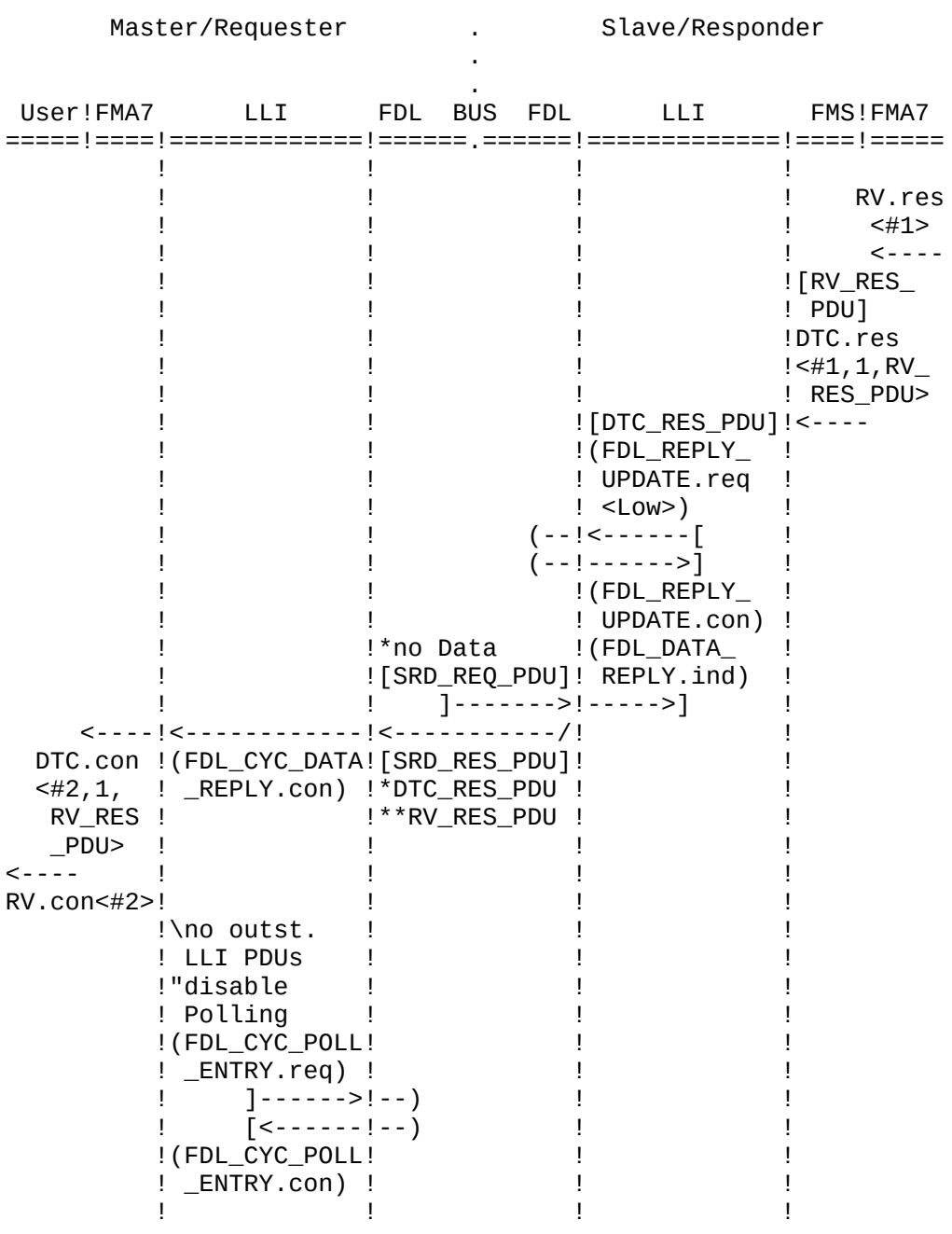


Figure 85. Master-Slave Communication Relationship / Connection for Acyclic Data Transfer with no Slave Initiative / Mapping of the Read Value Rem Service / Response part

Connection for Acyclic Data Transfer with Slave Initiative (MSAC_SI)

On a connection for Acyclic Data Transfer with Slave Initiative the mapping of all confirmed FMS services and of the unconfirmed FMS services (requester = master) is analogous to the mapping for a connection for Acyclic Data Transfer with no Slave Initiative (see data transfer definition).

The transmission of remote FMA7 services is not permitted on connections for Acyclic Data Transfer with Slave Initiative.

The polling (CSRD) is started for this communication relationship (FDL_CYC_POLL_ENTRY.req) during the connection establishment. The polling is continued until an Abort releases the connection.

Unconfirmed FMS services with requester = slave:

Unconfirmed FMS services are services which are not acknowledged by the user. For these services low or high priority may be used. The mapping of the unconfirmed FMS services onto Layer 2 is described here for the example of the InformationReport service. All other unconfirmed FMS services are mapped in the same way.

InformationReport service:

On a connection for Acyclic Data Transfer with Slave Initiative the Information-Report service is used for a single writing of data from a slave into a master. The FMS maps each request (InformationReport.req) onto the LLI service primitive DTA.req. The priority chosen by the user of the slave is transferred transparently. Corresponding to the priority, the LLI stores a DTA.req into the high or low priority Layer 2 update memory. The following SRD_REQ_PDU causes the update memory to be read. The SRD_RES_PDU transports the INFORMATION-REPORT_REQ_PDU to the master. The resulting FDL confirmation leads to a DTA.ind from the LLI of the master to FMS. FMS maps this DTA.ind onto a InformationReport.ind and passes it to the user.

If there is memory available again (Buffer_free) for the LLI of the master to receive another DTA_REQ_PDU, the LLI generates a DTA_ACK_PDU to signal this to the LLI of the slave. Corresponding to the priority with which the DTA_REQ_PDU was passed to LLI by the Layer 2, LLI maps the DTA_ACK_PDU either onto the Layer 2 service SRD with high priority or onto the Layer 2 service CSRD. The following SRD_REQ_PDU transports the DTA_ACK_PDU to the LLI of the slave. This does not lead to a confirmation to FMS.

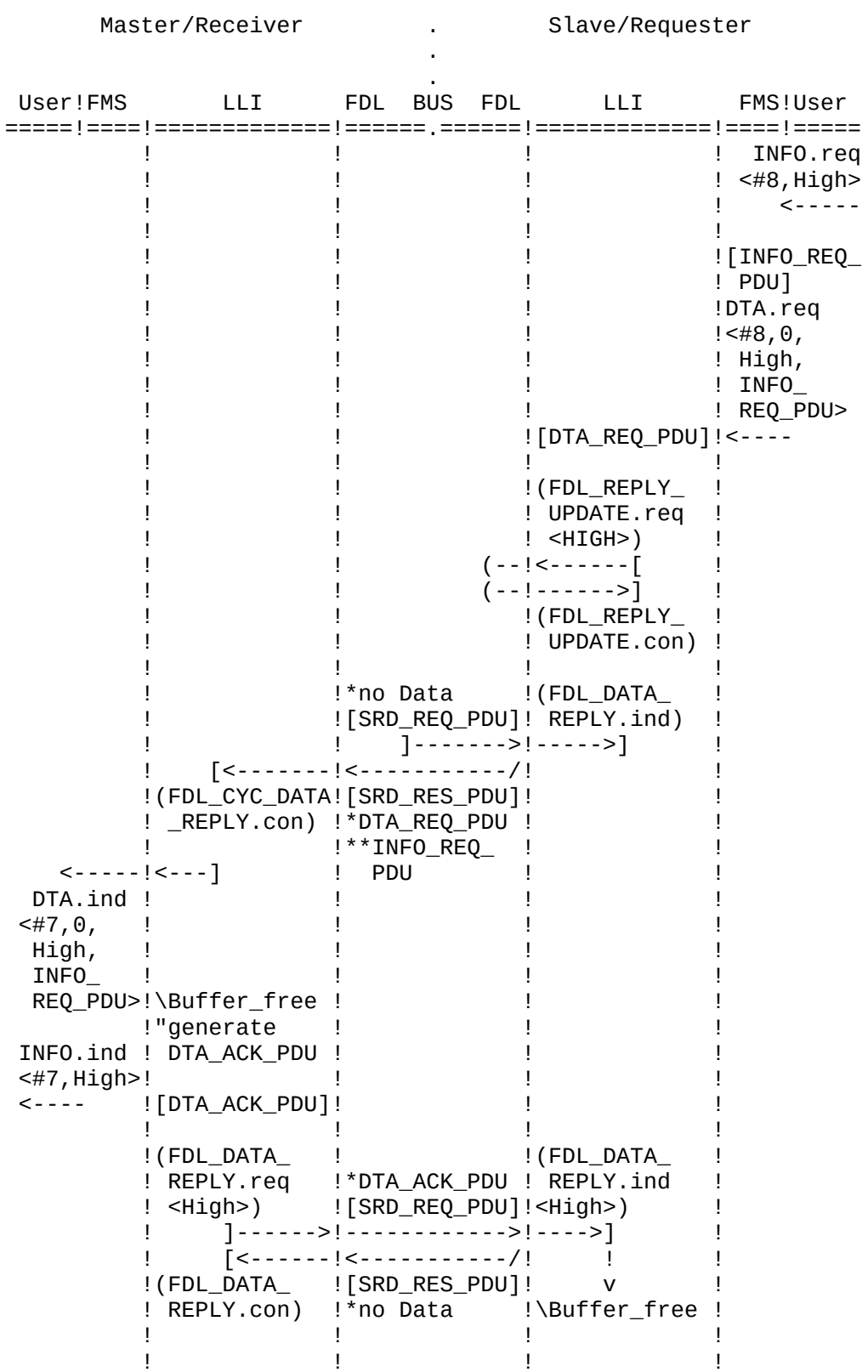


Figure 86. Master-Slave Communication Relationship / Connection for Acyclic Data Transfer with Slave Initiative / InformationReport with high Priority

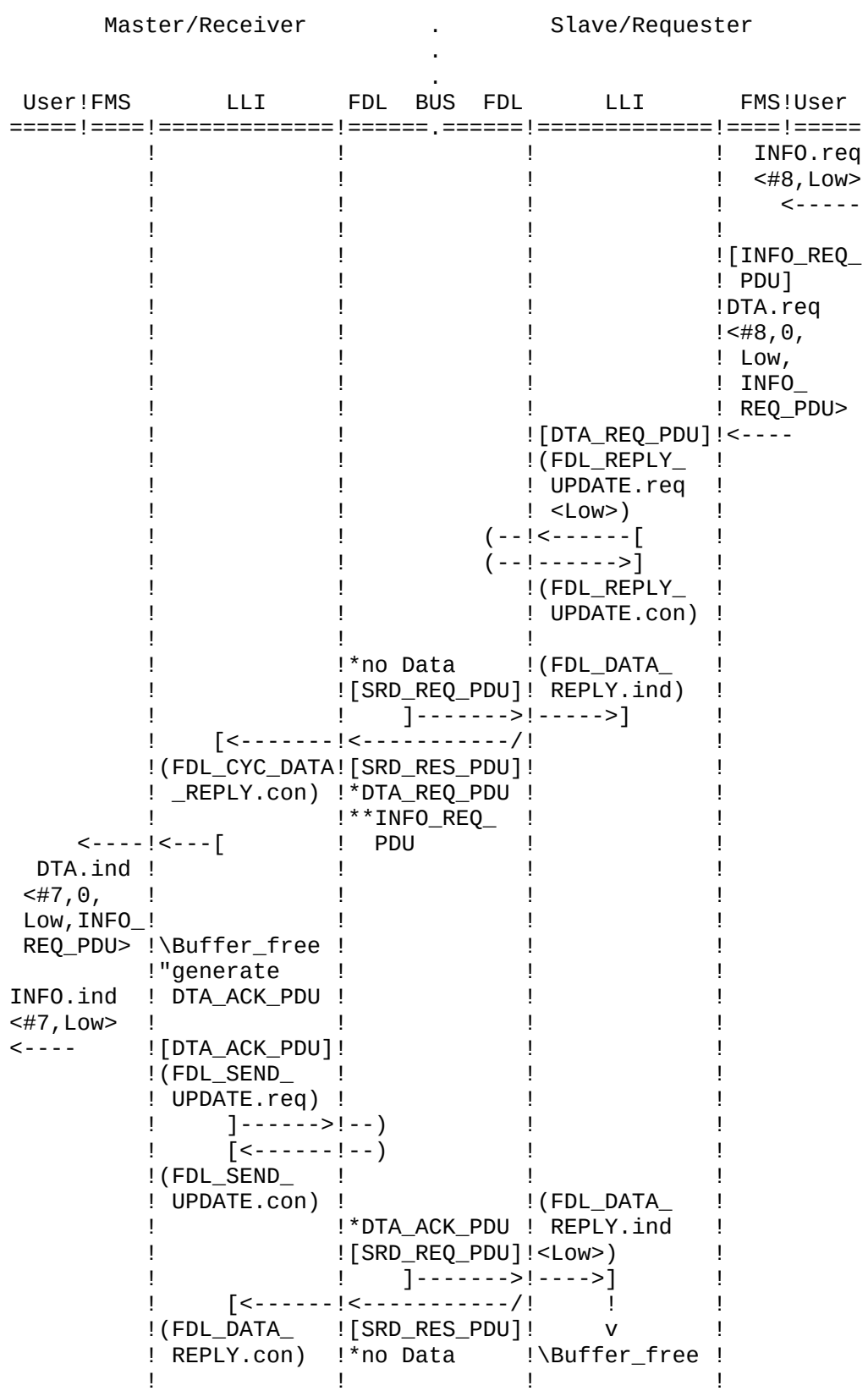


Figure 87. Master-Slave Communication Relationship / Connection for Acyclic Data Transfer with Slave Initiative / InformationReport with low Priority

6.3.4.3 Mapping of FMS/FMA7 Services onto Layer 2 for a Master-Master Communication Relationship

In this subclause the mapping of the FMS/FMA7 services onto the Layer 2 services is described. A master - master communication in Layer 2 is assumed.

Connection for acyclic Data Transfer (MMAC)

Connections for Acyclic Data Transfer shall be established like all other connection oriented communication relationships using a connection establishment service (Initiate).

During the subsequent data transfer phase only the Layer 2 service SDA is used. In this way parallel and mutual FMS services may be performed efficiently on the same connection. All confirmed and unconfirmed FMS services and all remote FMA7 services are permitted on this connection. Multiple connections for Acyclic Data Transfer may exist to a master at the same time. All these connections shall use different LSAPs. Optionally the connection may be monitored (see data transfer definition).

A connection for Acyclic Data Transfer is released with an Abort service (see connection release definition).

Confirmed FMS services:

Confirmed services are services which are acknowledged by the user. The mapping of the confirmed FMS services onto Layer 2 is described here for the example of the Read service. All other confirmed FMS services are mapped in the same way.

Read service:

On a connection for Acyclic Data Transfer the Read service is used for reading of data out of a master. The user marks each read request (Read.req) with a special identification (Invoke ID). In this way it is possible to relate the sent read request to the received read response. The FMS maps each read request onto the LLI service primitive DTC.req. The READ_REQ_PDU is transmitted from the requester to the responder with a low priority SDA_REQ_PDU. The resulting FDL_DATA_ACK.ind leads to a DTC.ind from the LLI of the responder to FMS. FMS maps the DTC.ind onto a Read.ind to the user.

When the user of the responder has passed the Read.res to FMS, FMS generates the READ_RES_PDU and passes it with a DTC.res to LLI. LLI passes the READ_RES_PDU to Layer 2 using a FDL_DATA_ACK.req <Low>. The resulting SDA_REQ_PDU transports the data to the requester. The READ_RES_PDU is passed to LLI using a FDL_DATA_ACK.ind. LLI passes it to FMS using a DTC.con. Then FMS passes the Read.con to the user.

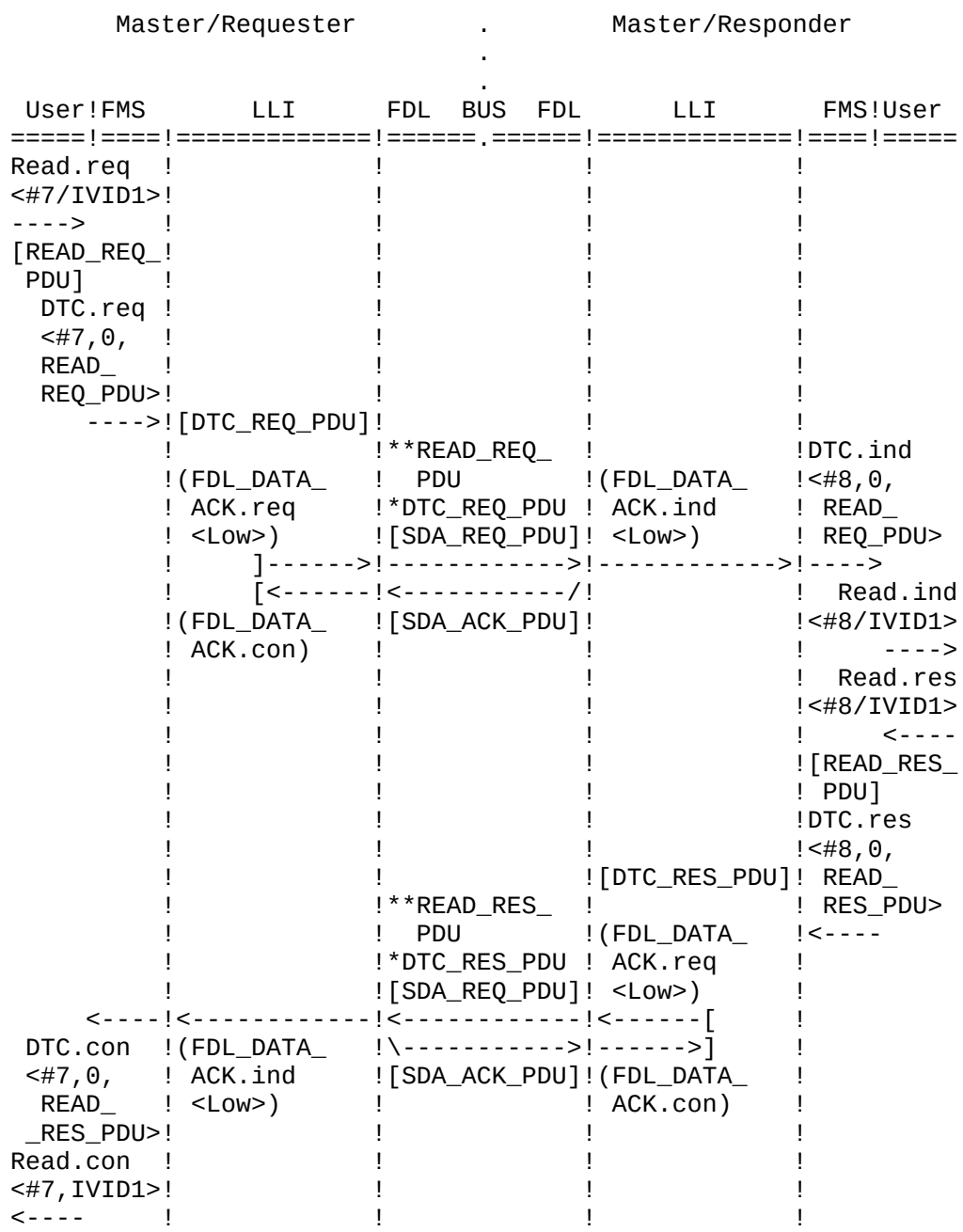


Figure 88. Master-Master Communication Relationship / Connection for Acyclic Data Transfer / Mapping of a confirmed Service for the Example of the Read Service of FMS

Unconfirmed FMS services:

Unconfirmed FMS services are services which are not acknowledged by the user. For these services low or high priority may be used. The mapping of the unconfirmed FMS services onto Layer 2 is described here for the example of the InformationReport service. All other unconfirmed FMS services are mapped in the same way.

InformationReport service:

On a connection for Acyclic Data Transfer the InformationReport service is used for a single writing of data into a master. The FMS maps each request (InformationReport.req) onto the LLI service primitive DTA.req. The priority chosen by the user is transferred transparently. LLI maps a DTA.req with high priority onto the Layer 2 service SDA with high priority. LLI maps a DTA.req with low priority onto the Layer 2 service SDA with low priority. Each request leads to a transmission of the INFORMATION-REPORT_REQ_PDU with a SDA_REQ_PDU from the requester to the receiver. The resulting FDL_DATA_ACK.ind leads to a DTA.ind from the LLI of the receiver to FMS. FMS maps the DTA.ind onto an InformationReport.ind to the user.

If there is memory available again (Buffer_free) for the LLI of the master (receiver) to receive another DTA_REQ_PDU, the LLI generates a DTA_ACK_PDU to signal this to the LLI of the master (requester). The LLI maps the DTA_ACK_PDU onto the Layer 2 service SDA with high or low priority corresponding to the priority with which the DTA_REQ_PDU was passed to LLI by the Layer 2. The SDA_REQ_PDU transports the DTA_ACK_PDU to the LLI of the master (requester). This does not lead to a confirmation to FMS.

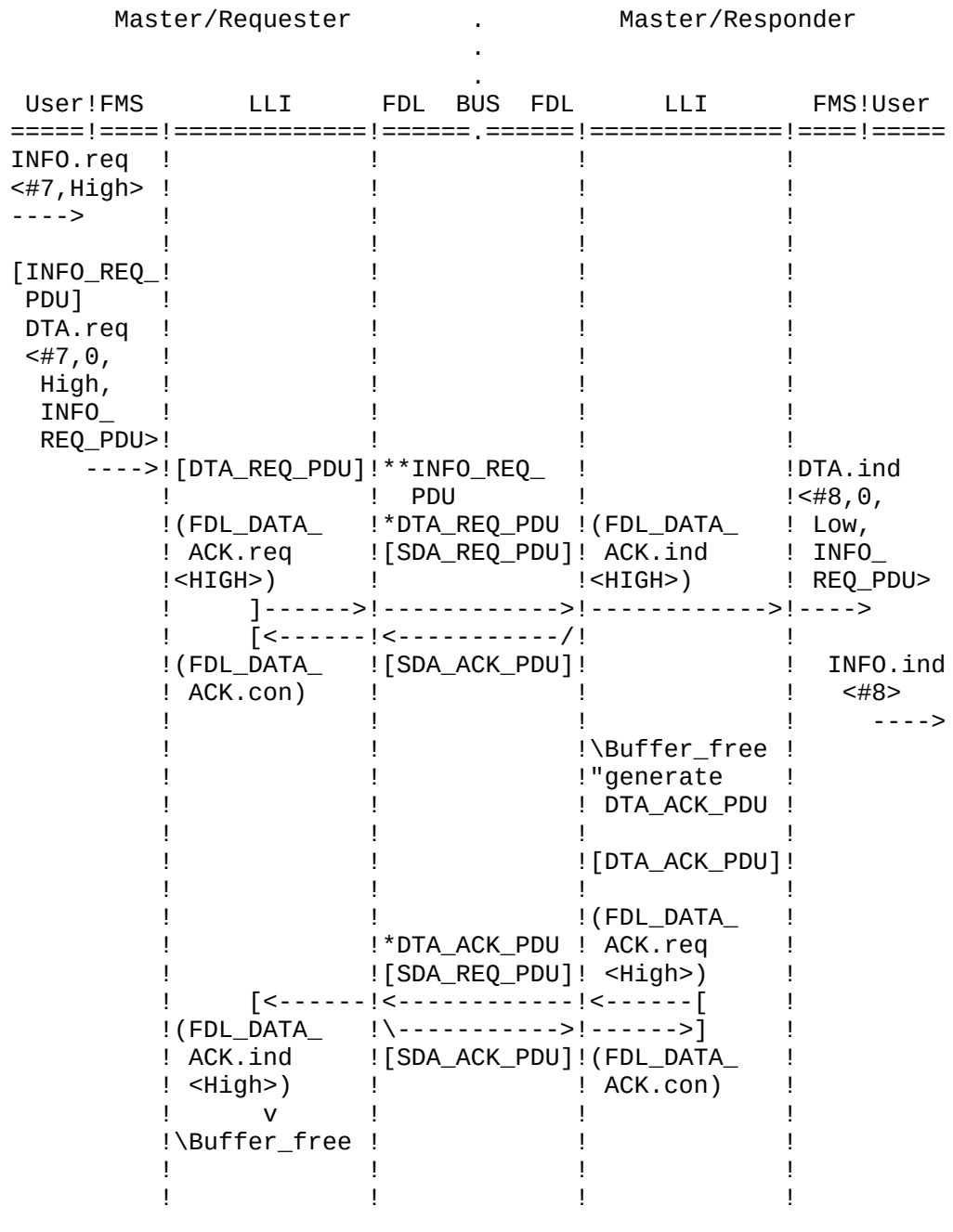


Figure 89. Master-Master Communication Relationship / Connection for Acyclic Data Transfer / InformationReport with high Priority

The sequence of an InformationReport with low priority is analogous to that shown the figure above. <Low> shall be read in place of <High>.

Remote FMA7 services:

All remote FMA7 services are acknowledged by the user. The mapping of these services onto Layer 2 is described here for the example of the Read Value Rem service. All other remote FMA7 services are mapped in the same way.

Read Value Rem service:

On a connection for Acyclic Data Transfer the Read Value Rem service is used for reading of variable values or counters out of a master. FMA7 maps each read request (ReadValueRem.req) onto the LLI service primitive DTC.req. The READ_VALUE-REM_REQ_PDU is transported with a SDA_REQ_PDU with low priority from the requester to the responder. The resulting FDL_DATA_ACK.ind leads to a DTC.ind from the LLI of the responder to FMA7. FMA7 maps the DTC.ind onto a ReadValueRem.ind to the user.

When the user of the responder has passed the Read-Value-Rem.res to FMA7, FMA7 generates the READ-VALUE-REM_RES_PDU and passes it to LLI with a DTC.res. LLI passes the READ-VALUE-REM_RES_PDU to Layer 2 with a FDL_DATA_ACK.req <Low>. The resulting SDA_REQ_PDU transports the data to the requester. The READ-VALUE-REM_RES_PDU is passed to LLI with the FDL_DATA_ACK.ind. LLI passes it to FMA7 using a DTC.con. Then FMA7 passes the ReadValueRem.con to the user.

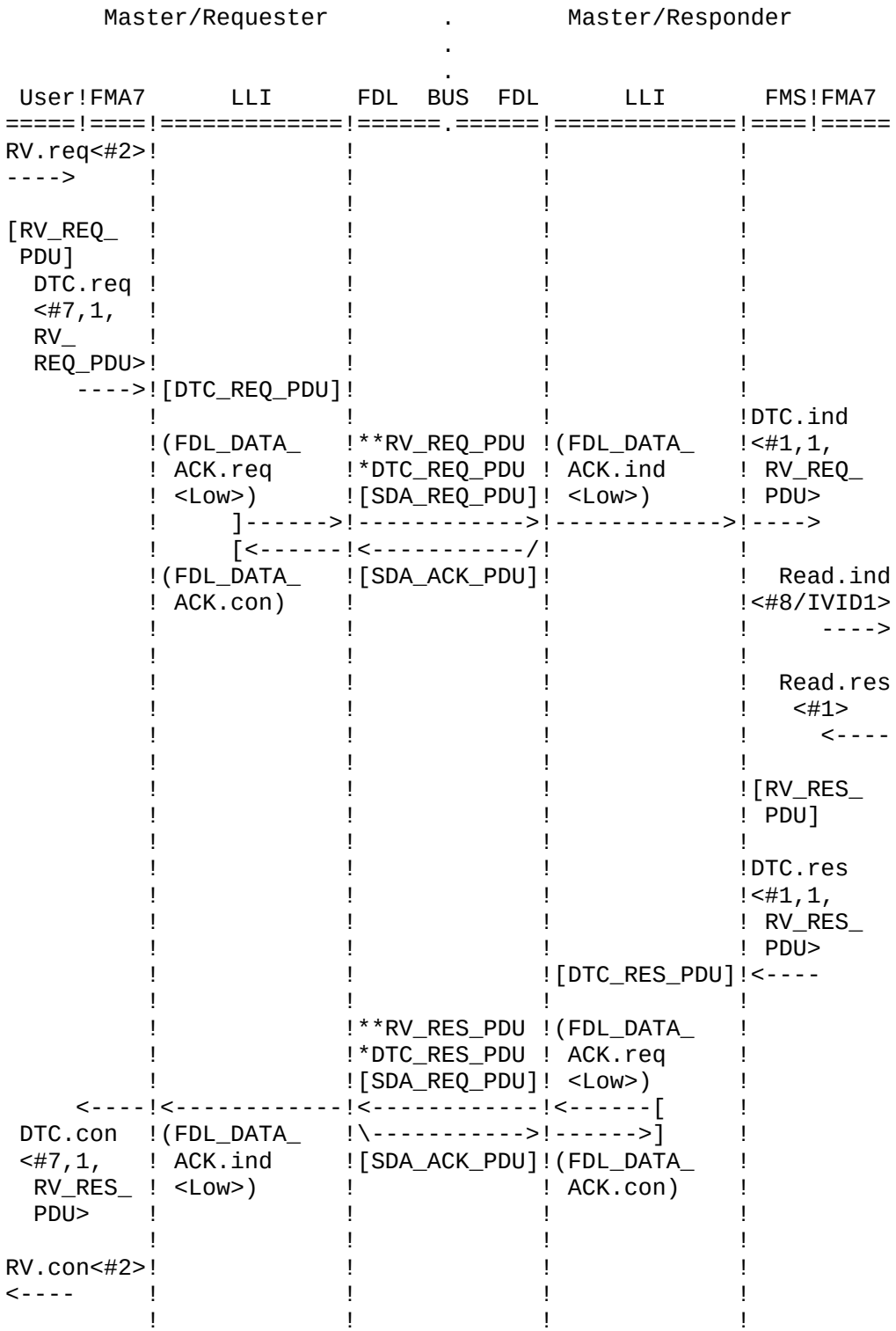


Figure 90. Master-Master Communication Relationship / Connection for Acyclic Data Transfer / Mapping of a remote FMA7 Service for the Example of the ReadValueRem Service

6.3.4.4 Monitoring on Data Transfer

Interpretation of the Layer 2 Confirmation Primitive

During the data transfer phase the parameter L_status of the Layer 2 confirmation primitive (FDL_XXX.con) is interpreted. Certain values of the parameter L_status (see formal LLI state machine definition) lead to a connection release and in some cases to an error message (LLI-FAULT.ind) to FMA7.

Monitoring the Connection

During the data transfer phase the availability of a connection may be monitored. According to the kind of connection two monitoring mechanisms are distinguished:

Monitoring of a connection for Acyclic Data Transfer:

This kind of monitoring is optional and specific to the connection. If a connection is to be monitored, both partners shall take part in monitoring. During the connection establishment phase this is guaranteed by the context test. The connection monitoring may be performed on master-master and master-slave connections (with or with no Slave Initiative). The monitoring mechanism is the same for master and slave on each acyclic connection type. The connection monitoring is configured specific to the connection by entering a time interval greater than 0 into the attribute ACI (Acyclic Control Interval, receive interval) in the context of the LLI CRL. ACI is the time interval in which the failure of a connection shall be surely recognized. The connection monitoring is started after a successful connection establishment (STimer, RTimer).

The correct operation of a connection for acyclic data transfer is recognized by receiving a correct LLI PDU during the monitoring interval ACI. It shall be guaranteed that both partners send a LLI PDU during the send interval (controlled by the STimer). If the send interval expires and no LLI PDU has been sent, then LLI sends a control message (IDLE_PDU). After each sending of a LLI PDU the send interval starts again. After each receipt of a LLI PDU the receive interval (controlled by the RTimer) starts again. This interval shall be equal for both partners. If no LLI PDU has been received during the receive interval, the connection is disturbed and shall be released (see connection release definition).

The send interval is a third of the receive interval (ACI).

Monitoring of a connection for Cyclic Data Transfer:

The monitoring is specific to the connection. It checks if during the time interval CCI (Cyclic Control Interval), which has been configured in the CRL, an event occurs, which confirms the correct operation of the connection.

The monitoring is started by the first confirmed Layer 7 request (in the master the first DTC.req from the FMS; in the slave the arrival of the first DTC_REQ_PDU). The receipt of an expected DTC_RES_PDU in the master, or the arrival of a SRD.ind in the slave, during the monitoring interval (timer T3) shows that the connection is operating correctly. In this case the LLI starts the monitoring interval again. If the monitoring interval expires, the connection is disturbed and shall be released (see see connection release definition).

The user of the master may monitor the user of the slave during the execution of the first request because it does not receive the response out of the local IDM. For all following requests with the same Invoke ID this is not possible because the response is taken from the IDM and the user cannot detect the actualization of the IDM (receipt of a DTC_RES_PDU). Because the actualization of the IDM during the connection monitoring is detected by LLI, LLI monitors automatically the user of the slave in addition to the connection. To guarantee the monitoring of the user at any time the connection monitoring is mandatory in the master.

In the slave the monitoring is optional. It only makes sense if the slave can react upon a connection failure (e.g. fail-safe, redundancy switch-over). The

failure of the user of the master cannot be detected with this monitoring.

6.3.4.5 Flow Control in LLI

The flow control regulates the LLI PDU traffic on connection-oriented communication relationships. The method is identical for connections for acyclic and cyclic data transfer. It includes confirmed and unconfirmed FMS services and all FMA7 services equally.

During the configuration of the CRL the resources for the confirmed LLI user services (max SCC, max RCC) and the unconfirmed services (max SAC, max RAC) are defined specific to the connection.

- max SCC contains the maximum permitted number of parallel confirmed LLI user services to the remote communication partner. For each confirmed service of the LLI user to be sent an own DTC requester state machine shall be available.
- max RCC contains the maximum permitted number of parallel confirmed LLI user services from the remote communication partner. For each confirmed service of the LLI user which is received an own DTC responder state machine shall be available.
- max SAC contains the maximum permitted number of parallel unconfirmed FMS services to the remote communication partner. For each unconfirmed FMS service which to be sent an own DTA requester state machine shall be available.
- max RAC contains the maximum permitted number of parallel unconfirmed FMS services from the remote communication partner. For each unconfirmed FMS service which is received an own DTA acknowledge state machine shall be available.

On connections for cyclic data transfer max SCC and max RCC shall have the value 0, because here no parallel confirmed services are permitted.

During connection establishment the resources of both communication partners are checked to be compatible (see context test in LLI definition).

In the data transfer phase LLI uses up to four counters on each connection (SCC, RCC, SAC, RAC) for the execution of parallel FMS services. LLI stores the number of currently used state machines (DTC requester, DTC responder, DTA requester and DTA acknowledge state machines) in these counters.

For the transmission of a confirmed LLI user service the LLI user passes a DTC.req to LLI. LLI uses a free DTC requester state machine and increments SCC. The arrival of the associated DTC_RES_PDU releases the DTC requester state machine and decrements SCC at the same time. If there is no free DTC requester state machine left for the DTC.req (SCC = max SCC), LLI performs a connection release.

Upon receipt of a confirmed LLI user service (DTC_REQ_PDU), LLI uses a free DTC responder state machine and increments RCC. The arrival of the associated DTC.res from the LLI user releases the DTC responder state machine and decrements RCC at the same time. If there is no free DTC responder state machine left upon receipt of a DTC_REQ_PDU (RCC = max RCC), LLI performs a connection release.

For the transmission of an unconfirmed FMS service on a connection FMS passes a DTA.req to LLI. LLI uses a free DTA requester state machine and increments SAC. The arrival of the associated DTA_ACK_PDU releases the DTA requester state machine and decrements SAC at the same time. If there is no free DTA requester state machine left for the DTA.req (SAC = max SAC), LLI ignores the request.

Upon receipt of an unconfirmed FMS service (DTA_REQ_PDU) the LLI uses a free DTA acknowledge state machine and increments RAC. If a free buffer is notified (Buffer_free) the DTA acknowledge state machine is released and RAC is decremented at the same time. If there is no free DTA acknowledge state machine left upon receipt of a DTA_REQ_PDU (RAC = max RAC), LLI performs a connection release.

6.4 Connectionless Communication Relationships

A connectionless communication relationship is either a one-to- many (multicast) or a one-to-all (broadcast) communication relationship. For connectionless communication relationships neither a connection establishment, nor a connection release is performed. They are always in the data transfer phase. Connectionless communication relationships are especially appropriate for application processes with the following requirements:

- Synchronization of local clocks
- Global alarms
- Synchronous start of process parts
- Snapshots of process objects

6.4.1 Broadcast Data Transfer

For the broadcast communication relationship the master sends an unconfirmed message to all other stations (masters and slaves).

Broadcast communication relationships have the following features:

- The requester of a broadcast message is always a master.
- The receivers of a broadcast message are all other stations with exception of the requester.
- Only the LLI service "DTU" (high or low priority) is permitted.
- Unconfirmed data transfer

Broadcast communication relationships use the following addressing in Layer 2:

- At the requester every LSAP (0, 2 to 62/NIL) may be used.
- At every receiver the global FDL address (127) and the global LSAP (63) shall be addressed.

6.4.2 Multicast Data Transfer

For a multicast communication relationship the master sends an unconfirmed message to a group of stations (masters and slaves).

Multicast communication relationships have the following features:

- The requester of a multicast message is always a master.
- The receiver of a multicast message is always a group of stations, which does not include the requester.
- Only the LLI service "DTU" (high or low priority) is permitted.
- Unconfirmed data transfer.

Multicast communication relationships use the following addressing in Layer 2:

- At the requester every LSAP (0, 2 to 62/NIL) may be used.
- At every receiver the global FDL address (127) and an individual LSAP (0, 2 to 62/NIL) shall be addressed.

6.4.3 Mapping of FMS Services onto Layer 2

In this clause the sequence of the data transfer with connectionless communication relationships is described for the example of the characteristic FMS service InformationReport. For a connectionless communication relationship only the unconfirmed FMS services InformationReport, UnsolicitedStatus and EventNotification are permitted. FMS maps these services on connectionless communication relationships onto the LLI service DTU. LLI maps the DTU service onto the Layer 2 service SDN. Only this Layer 2 service allows the simultaneous addressing of several stations.

For a multicast / broadcast communication relationship the InformationReport service shall be used to write data in several or in all other stations (masters and slaves).

Every user request (InformationReport.req) is mapped by FMS onto a DTU.req and results in a transmission of an INFO_REQ_PDU from the requester to the receivers using a SDN_REQ_PDU. The resulting FDL_DATA.con causes no action in the LLI of the requester. At the receivers, the FDL_DATA.ind causes the LLI to pass a DTU.ind to FMS. FMS constructs the InformationReport.ind for the user out of the DTU.ind.

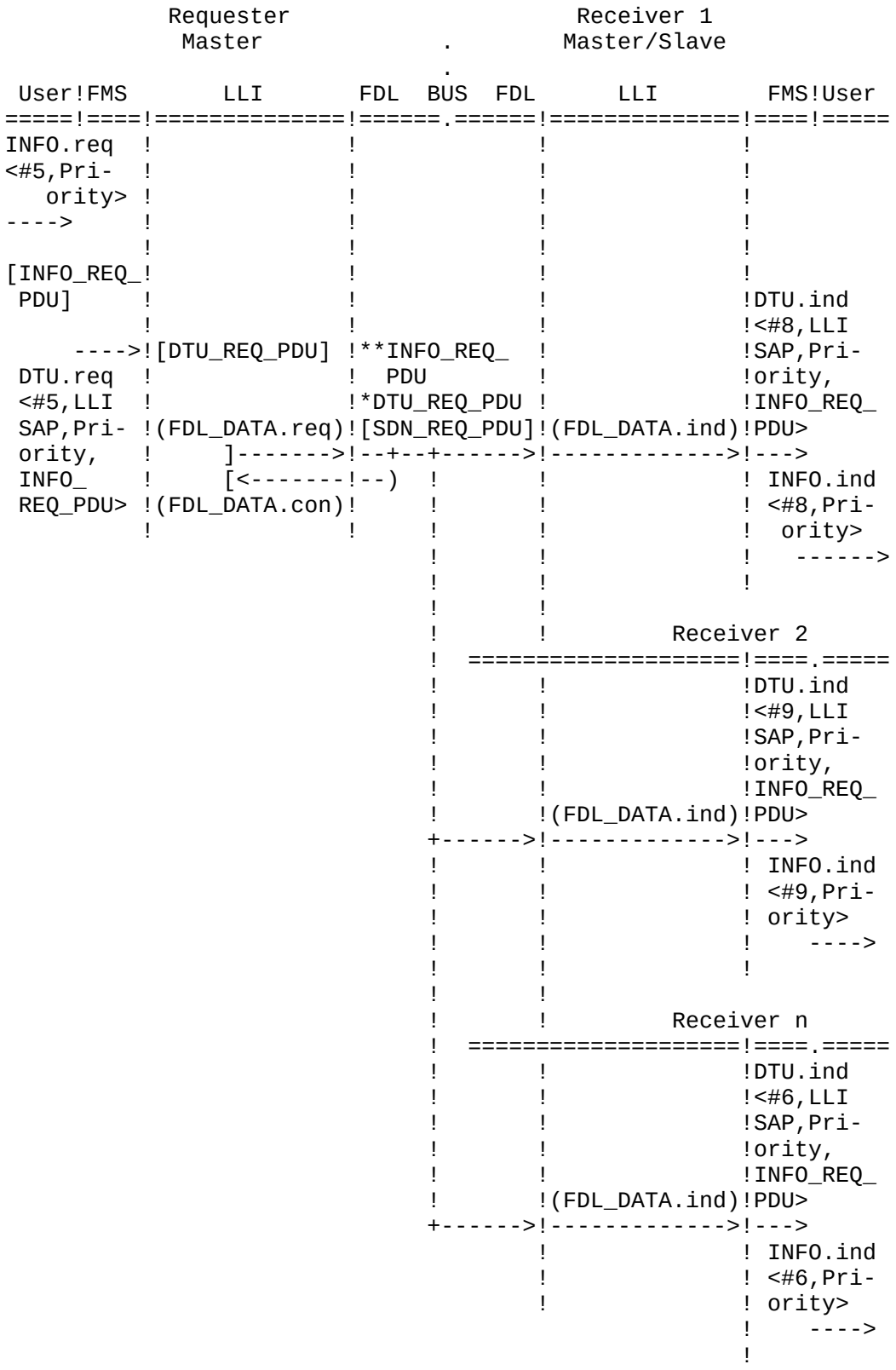


Figure 91. Broadcast Communication Relationship / Mapping of an unconfirmed FMS Service for the Example of InformationReport Service

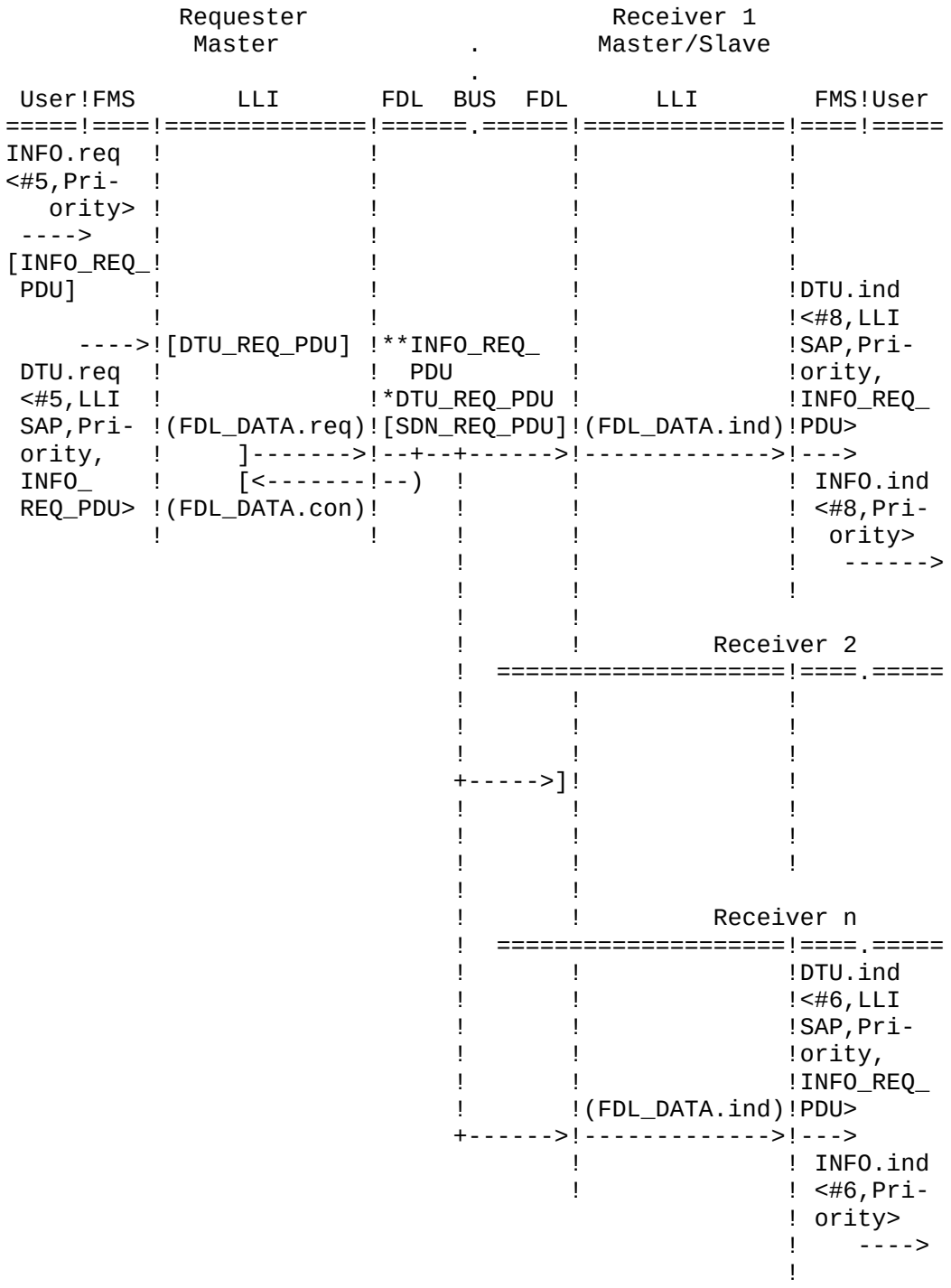


Figure 92. Multicast Communication Relationship / Mapping of an unconfirmed FMS Service for the Example of InformationReport Service / Receiver 2 is not member of this Multicast Station Group

6.5 LLI PDUs

LLI distinguishes between ten different PDU Types by unique function codes (FC). The assignment of function codes to PDU types is shown in the table below.

Table 20. Coding of the LLI PDU Types

FC	LLI PDU Type
0000	Associate Request PDU (ASS_REQ_PDU)
0001	Associate Response PDU (ASS_RES_PDU)
0010	Associate Negative Response PDU (ASS_NRS_PDU)
0011	Abort Request PDU (ABT_REQ_PDU)
0100	Data Transfer Confirmed Request PDU (DTC_REQ_PDU)
0101	Data Transfer Confirmed Response PDU (DTC_RES_PDU)
0110	Data Transfer Acknowledged Request PDU (DTA_REQ_PDU)
0111	Data Transfer Acknowledged Acknowledge PDU (DTA_ACK_PDU)
1000	Data Transfer Unconfirmed Request PDU (DTU_REQ_PDU)
1001	Idle Request PDU (IDLE_REQ_PDU)

Every LLI PDU consists of a number of octets. The significance of each bit within an octet is defined in the figure below.

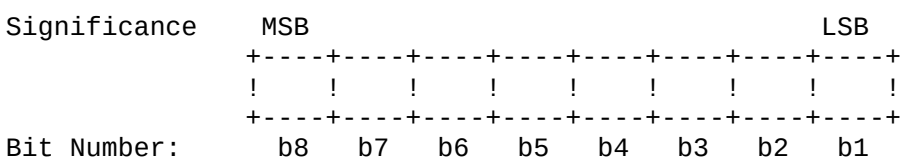


Figure 93. Significance of the bits within an octet

In the following PDU descriptions the octet 1 shall always be transmitted first. The general structure of a LLI PDU is shown in the following figure.

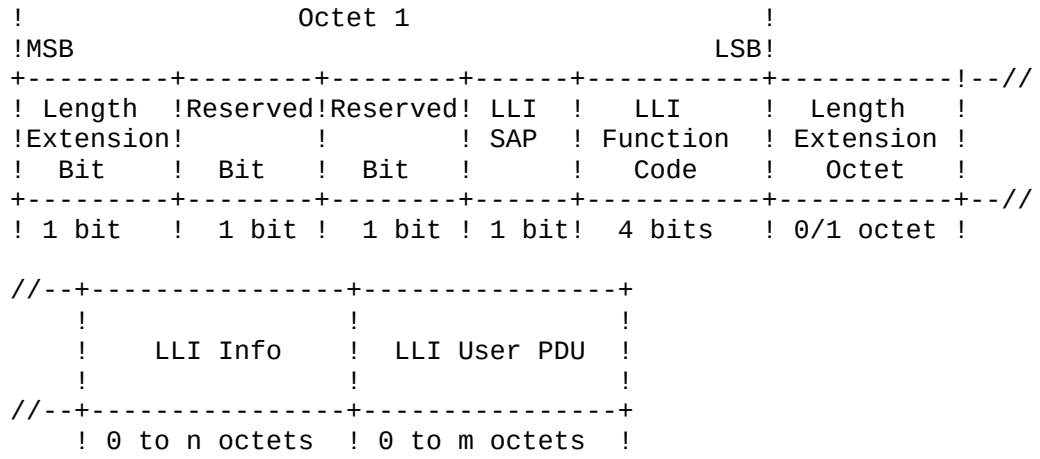


Figure 94. General Structure of a LLI PDU

Meaning of the fields of a LLI PDU:

Length Extension Bit

This field of length 1 bit shall indicate whether the fields Length Extension Octet and LLI Info are present. If the field Length Extension Bit has the value 0, then the fields Length Extension Octet and LLI Info are absent. If the field Length Extension Bit has the value 1, then the fields Length Extension Octet and LLI Info are present.

Reserved Bit

These fields each of length 1 bit are reserved for future use.

LLI SAP

This field of length 1 bit shall specify the LLI SAP which is configured in the CRL of the sending LLI for this communication relationship. The LLI SAP 0 is always assigned to the LLI user FMS. The LLI SAP 1 is always assigned to the LLI user FMA7.

LLI Function Code

This field shall contain the function code of the LLI PDU.

Length Extension Octet

This field gives the length of the field LLI Info in octets. The Length Extension Octet and the field LLI Info are only present if the Length Extension Bit has the value 1.

LLI Info

This field contains further LLI protocol information.

LLI User PDU

This field shall contain the PDU from the LLI user.

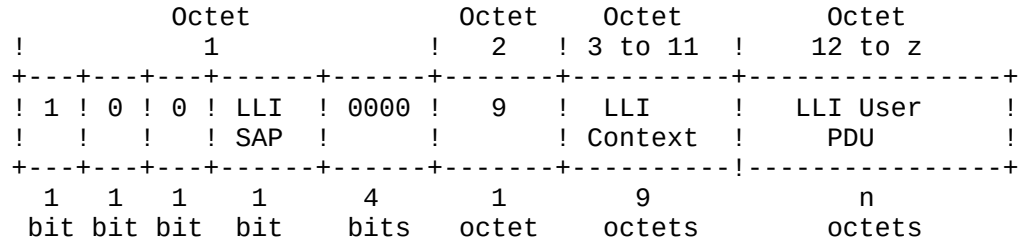


Figure 95. Structure of the ASS_REQ_PDU

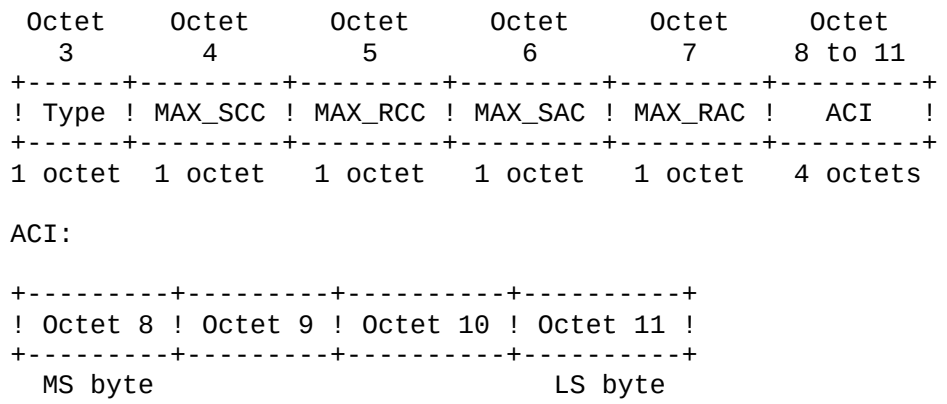


Figure 96. Structure of the Field LLI Context of the ASS_REQ_PDU

The field Type shall indicate the type of the communication relationship (see LLI CRL definition) and may contain the following values:

Table 21. Coding for the Field Type

b8 b1	<- bit position	
0 0 0 0 0 0 0 0	:	MMAC
0 0 0 0 0 0 0 1	:	MSAC
0 0 0 0 0 1 0 1	:	MSAC_SI
0 0 0 0 0 0 1 1	:	MSCY
0 0 0 0 0 1 1 1	:	MSCY_SI

The fields "MAX_SCC", "MAX_RCC", "MAX_SAC" and "MAX_RAC" may take the values 0 to 255 and shall be coded as Unsigned8.

The value of ACI shall be multiplied by 10 ms to determine the control interval for the Idle Control and shall be coded as Unsigned32.

(Range: 0 ms to (10 * (2**32 - 1)) ms).

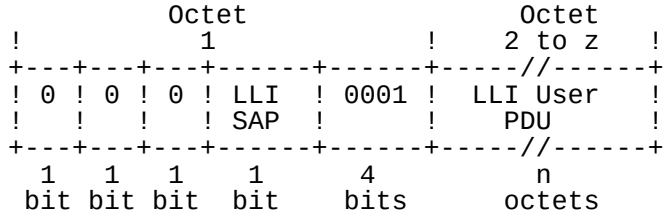


Figure 97. Structure of the ASS_RES_PDU

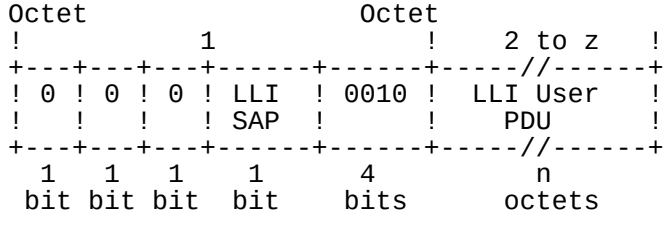


Figure 98. Structure of the ASS_NRS_PDU

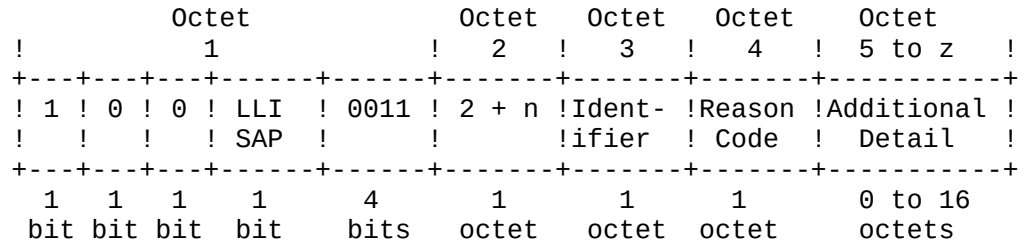


Figure 99. Structure of the ABT_REQ_PDU

Identifier

This field shall indicate the reason for the connection release and may contain the following values:

Table 22. Coding for the Field Identifier

b8 b1	<- bit position	
0 0 0 0 0 0 0 0	:	User
0 0 0 0 0 0 0 1	:	LLI User
0 0 0 0 0 0 1 0	:	LLI
0 0 0 0 0 0 1 1	:	Layer 2

Reason Code

This field shall indicate the reason for the connection release depending on the identifier.

Table 23. Coding for the Field Reason Code

If the reason lies in Layer 2, then the field Reason Code shall contain the L_status and may take the following values:

b8 b1	<- bit position	
0 0 0 0 0 0 0 0	:	RR
0 0 0 0 0 0 0 1	:	LR
0 0 0 0 0 0 1 0	:	NA
0 0 0 0 0 0 1 1	:	RDL
0 0 0 0 0 1 0 0	:	RDH

If the reason is located in LLI, then the field Reason Code may take the following values:

b8 b1	<- bit position	
0 0 0 0 0 0 0 0	:	ABT_RC1
0 0 0 0 0 0 0 1	:	ABT_RC2
0 0 0 0 0 0 1 0	:	ABT_RC3
to		to
0 0 0 0 1 0 1 1	:	ABT_RC12

If the reason is located at the LLI user or at the user, then the field Reason Code may take the following values:

b8 b1	<- bit position	
0 0 0 0 0 0 0 0	:	ABT_RC1
0 0 0 0 0 0 0 1	:	ABT_RC2
0 0 0 0 0 0 1 0	:	ABT_RC3
to		to
1 1 1 1 1 1 1 1	:	ABT_RC256

Additional Detail

This field shall contain additional details about the connection release depending on the identifier and Reason Code.

If the reason is located in Layer 2, then the field Additional Detail is omitted.

If the reason is located in LLI and the Reason Code indicates LLI-LLI context test negative, then the field Additional Detail shall contain the LLI context of the requester of the release (9 octets). For all other reason codes the field Additional Detail shall be omitted.

If the reason is located at the LLI user then the field Additional Detail shall contain the value (max. 16 octets) given by the LLI user.

If the reason is located at the user, then the field Additional Detail shall contain the value (max. 16 octets) given by the LLI User.

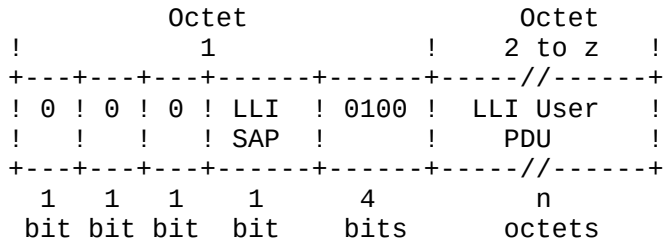


Figure 100. Structure of the DTC_REQ_PDU

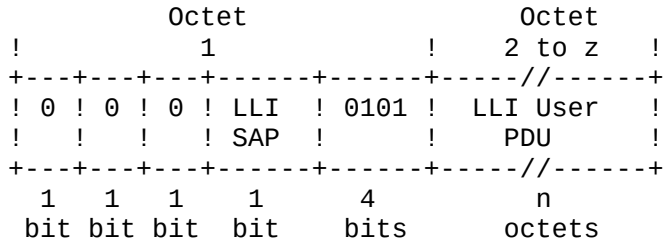


Figure 101. Structure of the DTC_RES_PDU

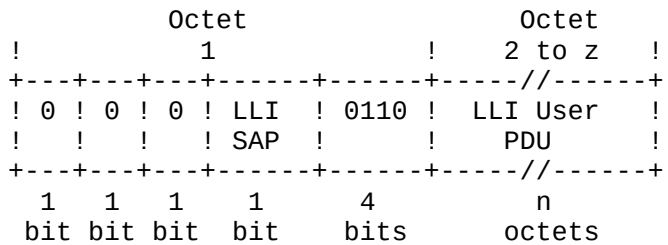


Figure 102. Structure of the DTA_REQ_PDU

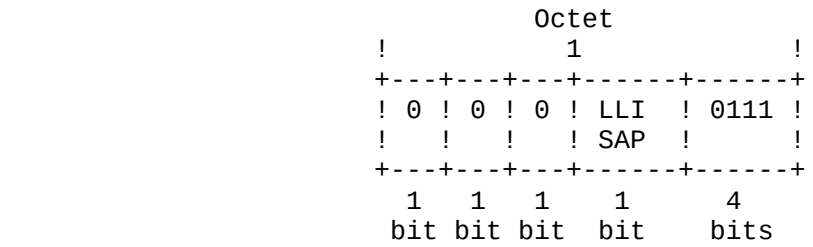


Figure 103. Structure of the DTA_ACK_PDU

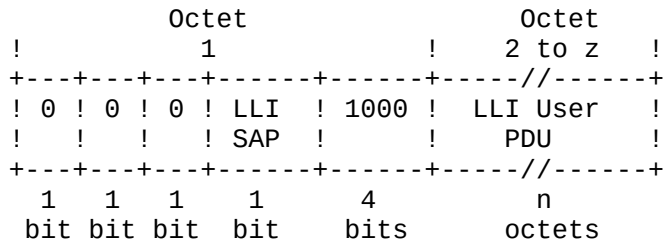


Figure 104. Structure of the DTU_REQ_PDU

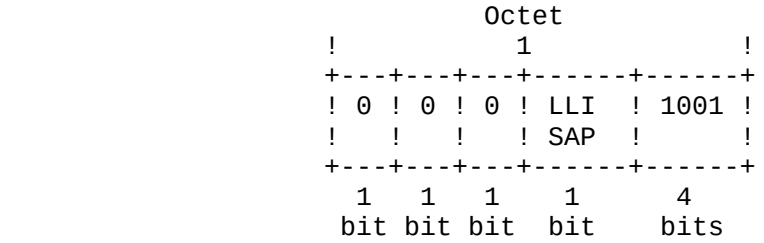


Figure 105. Structure of the IDLE_REQ_PDU

6.6 Start of LLI

After power-on, or after the LLI is reset from the FMA7, the start of the LLI begins. The starting phase is processed by the basic state machine of the LLI. Only after attaining the READY state is the LLI ready for operation and may accept requests for connection establishment or connectionless communication services (Broadcast / Multicast).

6.6.1 Conditions of Readiness for Operation

The following conditions shall be fulfilled for the successful termination of the starting phase (State of the LLI Basic State Machine = READY):

- Static part of the LLI CRL is present.
- Sufficient resources are available for the dynamic part of the LLI CRL.
- Sufficient resources are available for the Poll List and the LSAP Configurations.
- Sufficient resources are available for timers, buffers and state machines etc.

6.6.2 Predefinition of the dynamic part of the LLI CRL

In the starting phase the dynamic part of the LLI CRL is created and predefined by the LLI. For a connection-oriented communication relationship the predefinition is shown in the following figure. For a broadcast/multicast communication relationship the predefinition is shown in the second next figure.

	MANDATORY		Status		Closed
			Actual Remote Address		- -
			Actual Remote LSAP		- -
		Attributes for Administration of the Communication Relationship used internally by LLI	SCC		Null
			RCC		Null
			SAC		Null
			RAC		Null
			IMA		false
			Image Data (IDM)	Request Invoke ID FMS PDU	- -
			Poll Entry Enabled		false
			New		- -
			Old		- -

Figure 106. Predefinition of dynamic Part of the CRL for a Connection-oriented Communication Relationship

D	MANDATORY	Status	CONLS-REQUESTER/ CONLS-RECEIVER

Figure 107. Predefinition of dynamic Part of the CRL for a Broad/Multicast Communication Relationship

6.6.3 Generation of the Poll List and Transfer to Layer 2

In the starting phase the LLI creates the Poll List (see PROFIBUS Data Link Layer) depending on the master-slave communication relationships in the CRL. The multiplier value in the CRL shall be considered thereby for multiple entries.

The Poll List is delivered to Layer 2 with a FDL_CYC_DATA_REPLY.req service primitive. The necessary Service Access Point, the Poll List LSAP, shall be defined in the header of the LLI CRL.

6.6.4 Activation of Service Access Points of Layer 2

The Service Access Points of Layer 2 shall be activated in the start-up phase by LLI. The LLI of the master calls the service SAP Activate FMA1/2 for all LSAPs. The LLI of the slave calls for all RSAPs the service RSAP Activate FMA1/2 and for all LSAPs the service SAP Activate FMA1/2.

The following conditions apply in setting the LSAP parameters:

At the master

- For every master - master communication relationship for acyclic data transfer with the connection attribute = "D or O" which is registered in the LLI CRL, a LSAP with the following parameters shall be activated:
  Access: Rem_add, if connection attribute = "D"
  All, if connection attribute = "O"
  Service: SDA, Role_in_service = Both
  The values of the other parameters result from the entries of the master - master communication relationships in the static part of the LLI CRL.

- For master - master communication relationships for acyclic data transfer with the connection attribute = "I", the assigned LSAP of the Layer 2 shall not be activated in the start-up phase of LLI.

- All master - slave communication relationships shall be registered in the Poll List of Layer 2. The Poll List LSAP of Layer 2 shall be activated with the following parameters:
  Access: All
  Service: CSRD, Role_in_service = Initiator
  SRD, Role_in_service = Initiator, if for a connection the max PDU length for high priority messages is configured to be larger than 0.
  Confirm_mode: Data
  The values of the other parameters result from the entries of the master - slave communication relationships in the static part of the LLI CRL.

- For all broadcast or multicast communication relationships as a requester an individual LSAP shall be activated with the following parameters:
  Access: All
  Service: SDN, Role_in_service = Initiator
  The values of the other parameters results from the entries of the broadcast- or multicast communication relationships in the static part of the LLI CRL.

- For all broadcast communication relationships as a receiver the LSAP 63 shall be activated with the following parameters:

  Access: All

  Service: SDN, Role_in_service = Responder

  The values of the other parameters result from the entries of the broadcast communication relationships in the static part of the LLI CRL.
- For every multicast communication relationship as a receiver the assigned LSAP shall be activated with the following parameters:

  Access: All

  Service: SDN, Role_in_service = Responder

  The values of the other parameters result from the entries of the multicast communication relationships in the static part of the LLI CRL.

At the slave

- For every communication relationship of type MSAC or MSCY which is registered in the CRL, the assigned RSAP shall be activated with the service RSAP Activate FMA1/2 with the following parameters:

  Access: Rem_add, if connection attribute = "D"
  All, if connection attribute = "O"

  Indication_mode: Data

  The values of the other parameters result from the entries of the master - slave communication relationships in the static part of the LLI CRL.
- For all broadcast communication relationships as a receiver, the LSAP 63 shall be activated with the following parameters:

  Access: All

  Service: SDN, Role_in_service = Responder

  The values of the other parameters result from the entries of the broadcast communication relationships in the static part of the LLI CRL.
- For every multicast communication relationship as a receiver, the assigned LSAP shall be activated with the following parameters:

  Access: All

  Service: SDN, Role_in_service = Responder

  The values of the other parameters result from the entries of the multicast communication relationships in the static part of the LLI CRL.

6.7 Formal Description of the LLI State Machines

The formal description of LLI is based upon a model. It describes the LLI protocol with the help of state machines (represented by state diagrams). The protocol sequences are described by different states (represented by ellipses with the name of the state) and transitions between states (represented by lines with arrows). State changes are caused by events (for example time out) combined with conditions. The state changes in turn lead to actions or reactions.

Three different kinds of state machines are distinguished between:

a) the basic state machine of the LLI (start)

b) the CREF related state machines (connection establishment, open, connection release, DTU)

c) the service related state machines (DTC, DTCC, DTA, IDLE)

The description of the state transitions is similar to the draft ISO 8802 Part 4. The elements used are the sequence (current state, event / condition => action, next state) as well as constants, variables, service primitives, functions and procedures.

The detailed description of the state transitions and actions is structured for every state change as follows:

The first line defines the current state, the name of the transition and the next state. Below, in the subsequent lines, shall follow:

a) the events and conditions, which shall have become true for the transition to the following state, and after that:

b) the actions which are executed before entering the next state.

The events and conditions to be evaluated and the actions to be executed are described with a syntax based on the programming language PL/1. The constants, variables, functions and procedures are described similarly to the data definitions in the programming language PL/1. The PL/1 based descriptions are used to simplify the understanding of the written text. They do not intend to represent compatible elements of the language PL/1.

The FDL indications and confirmations shall represent events for the LLI state machines. The assignment of these primitives to the state machines is dependent on the implementation and is not specified in this specification.

Performance optimized slave implementations, which are able to issue the immediate response of Layer 7 with the immediate response of Layer 2, do not need to realize explicitly certain states and state transitions in the related state machines of the responder, as long as the specified functionality is assured.

Definition of an unllowed PDU or an unallowed service primitive:

The PDU or the service primitive is defined in this specification and used in the implementation, but is not allowed in the current state.

Definition of an unknown PDU or an unknown service primitive:

The PDU or the service primitive is either not defined in this specification or not used in the implementation.

All incoming events are processed first by the service related state machines (DTC, DTCC, DTA, IDLE). If the current event has no impact on these four state machines or if they are not existing, it shall be processed by the CREF related state machines (open, connection establishment, connection release, DTU). If the current event has no impact on these four state machines or if they are not existing, it shall be processed by the basic state machine. If one of the following FDL service primitives ((C)SRD.con/SRD.ind)has an impact on two LLI state machines it shall be processed by both state machines.

In the following figures, rectangles with numbers are used to mark transitions into another state machine.

Legend used in the state diagrams:

italic:	Actions
normal:	Events or Commands
	Transition for all Connection Types
	Transition only for Master-Slave Connections with and with no Slave Initiative
	Transition only for Master-Slave Connections with no Slave Initiative
	Transition only for Master-Master Connections
D:	Defined Connection, Master-Master or Master-Slave Communication
I:	Open Connection at the Initiator, Master-Master Communication
O:	Open Responder SAP
(M)	Transition only for the Master of a Master-Slave Connection
(S)	Transition only for the Slave of a Master-Slave Connection

6.7.1 Start of LLI

State Diagram for Start of LLI

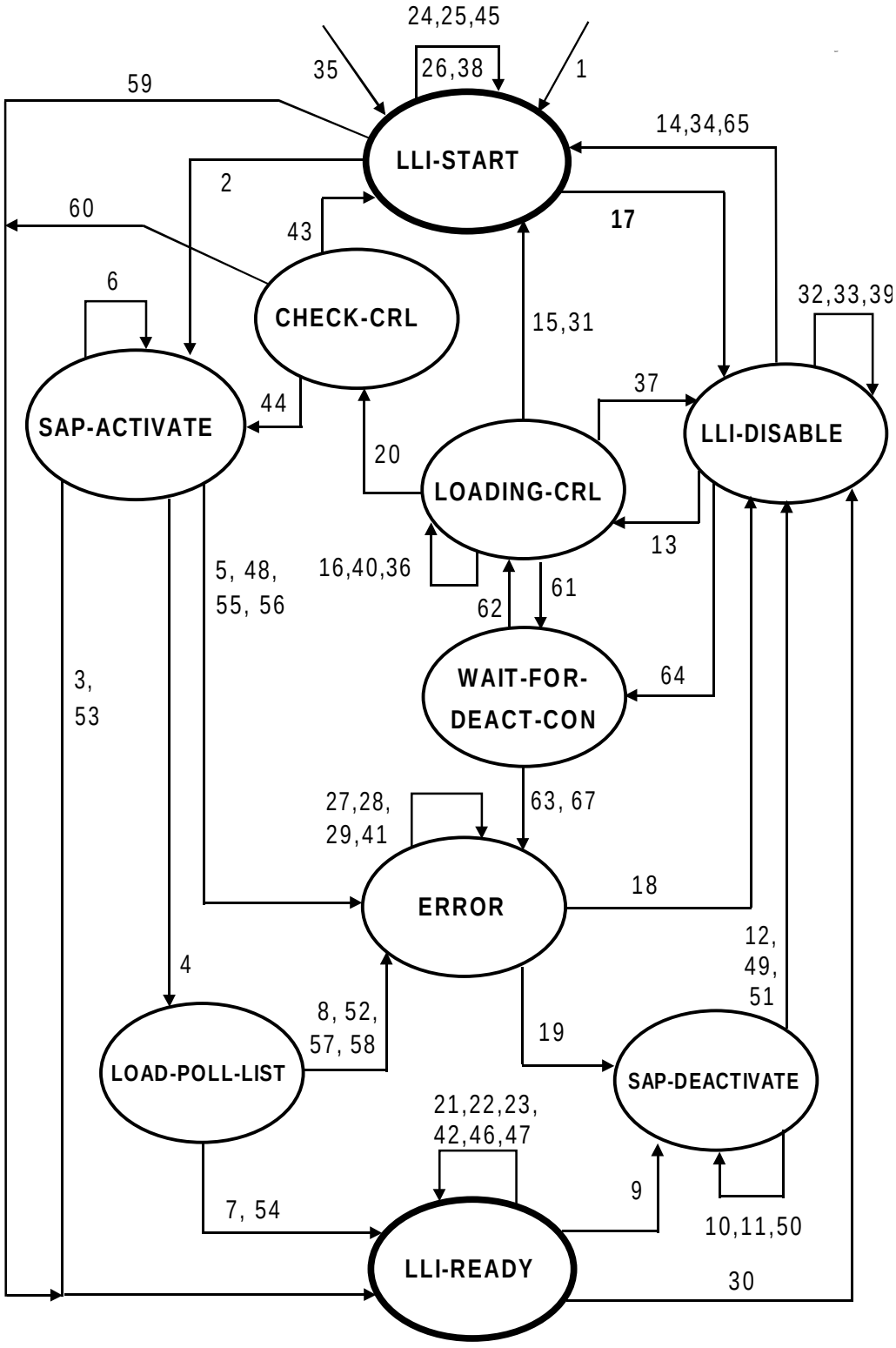


Figure 108. State Diagram for Start of LLI

1 Description of State Transitions for start

Start

Current State Event \Exit Condition => Action Taken	Transition	Next State
Power-On	**1**	**LLI-START**
=> start LLI CRL check CRL load flag := false		
LLI-START	**2**	**SAP-ACTIVATE**
LLI CRL check finished \valid LLI CRL available AND sufficient resources AND LLI CRL contains at least one communication relationship with a connection attribute unequal ”I” OR no connection attribute => predefine the dynamic part of the LLI CRL (R)SAP_ACT.req		
LLI-START	**59**	**LLI-READY**
LLI CRL check finished \valid LLI CRL available AND sufficient resources AND LLI CRL contains only communication relationships with a connection attribute equal ”I” => predefine the dynamic part of the LLI CRL (R)SAP_ACT.req		
SAP-ACTIVATE	**3**	**LLI-READY**
(R)SAP_ACT.con(OK) \all SAPs activated AND LLI CRL contains no Poll List LSAP AND CRL load flag = false => start of all LLI state machines related to CREF		
SAP-ACTIVATE	**53**	**LLI-READY**
(R)SAP_ACT.con(OK) \all SAPs activated AND LLI CRL contains no Poll List LSAP AND CRL load flag = true => start of all LLI state machines related to CREF LLI Enable.con(OK)		
SAP-ACTIVATE	**4**	**LOAD-POLL-LIST**
(R)SAP_ACT.con(OK) \LLI CRL contains one Poll List LSAP AND all SAPs activated => CSRD.req(Poll_List)		
SAP-ACTIVATE	**5**	**ERROR**
(R)SAP_ACT.con(NO/IV) \CRL load flag = false => LLI-Fault.ind<RC = LLI_FMA7_RC1, AD = M_status>		
SAP-ACTIVATE	**55**	**ERROR**
(R)SAP_ACT.con(NO/IV) \CRL load flag = true => LLI-Fault.ind<RC = LLI_FMA7_RC1, AD = M_status> LLI Enable.con(IV)		

Start

Current State Event \Exit Condition => Action Taken	Transition	Next State
SAP-ACTIVATE (R)SAP_ACT.con(OK) \not all SAPs activated => (R)SAP_ACT.req, do not activate LSAP 1 if it is already activated	**6**	**SAP-ACTIVATE**
LOAD-POLL-LIST CSRD.con(L_status = OK) \CRL load flag = false => start of all LLI state machines related to CREF	**7**	**LLI-READY**
LOAD-POLL-LIST CSRD.con(L_status = OK) \CRL load flag = true => start of all LLI state machines related to CREF LLI Enable.con (OK)	**54**	**LLI-READY**
LOAD-POLL-LIST CSRD.con(L_status = NO/LS/LR/IV) \CRL load flag = false => LLI-Fault.ind<RC = LLI_FMA7_RC17, AD = L_status>	**8**	**ERROR**
LOAD-POLL-LIST CSRD.con(L_status = NO/LS/LR/IV) \CRL load flag = true => LLI-Fault.ind<RC = LLI_FMA7_RC17, AD = L_status> LLI Enable.con(IV)	**57**	**ERROR**
LLI-READY LLI Disable.req \LLI CRL contains communication relationship with CREF > 1 => release all established connections and currently establishing connections, except the default management connection, <RC = ABT_RC24>; delete all LLI state machines related to CREF except for the default management CREF machines SAP_DEACT.req, CRL load flag := true	**9**	**SAP-DEACTIVATE**
SAP-DEACTIVATE SAP_DEACT.con(OK) \SAP_DEACT.req not yet called for all SAPs in {0, 2 to 63, NIL} => SAP_DEACT.req	**10**	**SAP-DEACTIVATE**
SAP-DEACTIVATE SAP_DEACT.con(M_status = NO/IV) \SAP_DEACT.req not yet called for all SAPs in {0, 2 to 63, NIL} => LLI-Fault.ind<RC = LLI_FMA7_RC2, AD = M_status> SAP_DEACT.req	**11**	**SAP-DEACTIVATE**

Start

Current State Event \Exit Condition => Action Taken	Transition	Next State
SAP-DEACTIVATE SAP_DEACT.con(OK) \SAP_DEACT.req called for all SAPs in {0, 2 to 63, NIL} => LLI Disable.con	**12**	**LLI-DISABLE**
LLI-DISABLE LLI Load CRL.req \LLI CRL entry OK AND Number of LLI CRL Entries not equal 0 => LLI Load CRL.con(OK)	**13**	**LOADING-CRL**
LLI-DISABLE LLI Load CRL.req \LLI CRL entry OK AND Number of LLI CRL Entries equals 0 AND CREF 1 existing in current CRL AND state of CREF 1 equals closed => SAP_DEACT.req(SSAP = 1)	**64**	**WAIT-FOR-DEACT-CON**
LLI-DISABLE LLI Load CRL.req \LLI CRL entry OK AND Number of LLI CRL Entries equals 0 AND CREF 1 existing in current CRL AND state of CREF 1 unequals closed => LLI Load CRL.con (SC)	**65**	**LLI-START**
LLI-DISABLE LLI Load CRL.req \LLI CRL entry NOK => LLI Load CRL.con(IV)	**14**	**LLI-START**
LOADING-CRL LLI Load CRL.req \LLI CRL entry NOK => LLI Load CRL.con(IV)	**15**	**LLI-START**
LOADING-CRL LLI Load CRL.req \LLI CRL entry OK AND (LLI CRL entry equals not CREF 1 OR (LLI CRL entry equals CREF 1 AND CREF 1 not existing in current CRL)) => LLI Load CRL.con(OK)	**16**	**LOADING-CRL**
LOADING-CRL LLI Load CRL.req \LLI CRL entry OK AND (LLI CRL entry equals CREF 1 AND CREF 1 existing in current CRL AND state of CREF 1 equals closed) => SAP_DEACT.req(SSAP = 1)	**61**	**WAIT-FOR-DEACT-CON**

Start

Current State Event \Exit Condition => Action Taken	Transition	Next State
LOADING-CRL LLI Load CRL.req \LLI CRL entry OK AND (LLI CRL entry equals CREF 1 AND CREF 1 existing in current CRL AND state of CREF 1 unequals closed) => LLI Load CRL.con (SC)	**66**	**LLI-START**
WAIT-FOR-DEACT-CON SAP_DEACT.con(OK) => LLI Load CRL.con(OK)	**62**	**LOADING-CRL**
WAIT-FOR-DEACT-CON SAP_DEACT.con(NO/IV) => LLI Load CRL.con(SC) LLI_Fault.ind< RC = LLI_FMA7_RC2, AD = M_status>	**63**	**ERROR**
WAIT-FOR-DEACT-CON unallowed FMA1/2 primitive => LLI Load CRL.con(SC) LLI_Fault.ind< RC = LLI_FMA7_RC22, AD = code of the primitive>	**67**	**ERROR**
LLI-START LLI Disable.req => LLI Disable.con CRL load flag := true	**17**	**LLI-DISABLE**
ERROR LLI Disable.req \no SAP activated in {0, 2 to 63, NIL} => LLI Disable.con CRL load flag := true	**18**	**LLI-DISABLE**
ERROR LLI Disable.req \activated SAPs in {0, 2 to 63, NIL} available => SAP_DEACT.req CRL load flag := true	**19**	**SAP-DEACTIVATE**
LOADING-CRL LLI Enable.req => start LLI CRL check	**20**	**CHECK-CRL**
LLI-READY any LLI service.req to the LLI FMA7 interface \NOT LLI reset.req AND NOT LLI Disable.req AND NOT LLI Load CRL.req AND NOT LLI Enable.req => execute service service.con(+/-) to FMA7	**21**	**LLI-READY**
LLI-READY LLI Enable.req => LLI Enable.con(SC)	**22**	**LLI-READY**

Start

Current State Event \Exit Condition => Action Taken	Transition	Next State
LLI-READY LLI Load CRL.req => LLI Load CRL.con(SC)	**23**	**LLI-READY**
LLI-START any LLI service.req to the LLI FMA7 interface \NOT LLI Disable.req AND NOT LLI Load CRL.req AND NOT LLI Enable.req AND NOT LLI Reset.req => execute service service.con(+/-) to FMA7	**24**	**LLI-START**
LLI-START LLI Enable.req => LLI Enable.con(SC)	**25**	**LLI-START**
LLI-START LLI Load CRL.req => LLI Load CRL.con(SC)	**26**	**LLI-START**
ERROR any service.req to the LLI FMA7 interface \NOT LLI reset.req AND NOT LLI Disable.req AND NOT LLI Load CRL.req AND NOT LLI Enable.req => execute service service.con(+/-) to FMA7	**27**	**ERROR**
ERROR LLI Enable.req => LLI Enable.con(SC)	**28**	**ERROR**
ERROR LLI Load CRL.req => LLI Load CRL.con(SC)	**29**	**ERROR**
LLI-READY LLI Disable.req \LLI CRL contains only CREF 1 (Management CREF) => LLI Disable.con CRL load flag := true	**30**	**LLI-DISABLE**
LLI-DISABLE any service.req to the LLI FMA7 interface \NOT LLI reset.req AND NOT LLI Disable.req AND NOT LLI Load CRL.req AND NOT LLI Enable.req => execute service service.con(+/-) to FMA7	**32**	**LLI-DISABLE**

Start

Current State Event \Exit Condition => Action Taken	Transition	Next State
LLI-DISABLE LLI Disable.req => LLI Disable.con CRL load flag := true	**33**	**LLI-DISABLE**
LLI-DISABLE LLI Enable.req => LLI Enable.con(SC)	**34**	**LLI-START**
<any State> LLI reset.req => LLI reset.con start LLI CRL check CRL load flag := false reset all CREFs	**35**	**LLI-START**
LOADING-CRL LLI service.req to the LLI FMA7 interface \NOT LLI reset.req AND NOT LLI Disable.req AND NOT LLI Enable.req AND NOT LLI Load CRL.req => execute service service.con(+/-) to FMA7	**36**	**LOADING-CRL**
LOADING-CRL LLI Disable.req => LLI Disable.con CRL load flag := true	**37**	**LLI-DISABLE**
LLI-START any service primitive to the LLI FMA7 interface \service not supported OR unknown service primitive => LLI-Fault.ind <RC = LLI_FMA7_RC9>	**38**	**LLI-START**
LLI-DISABLE any service primitive to the LLI FMA7 interface \service not supported OR unknown service primitive => LLI-Fault.ind <RC = LLI_FMA7_RC9>	**39**	**LLI-DISABLE**
LOADING-CRL any service primitive to the LLI FMA7 interface \service not supported OR unknown service primitive => LLI-Fault.ind <RC = LLI_FMA7_RC9>	**40**	**LOADING-CRL**
ERROR any service primitive to the LLI FMA7 interface \service not supported OR unknown service primitive => LLI-Fault.ind <RC = LLI_FMA7_RC9>	**41**	**ERROR**

Start

Current State Event \Exit Condition => Action Taken	Transition	Next State
LLI-READY any service primitive to the LLI FMA7 interface \service not supported OR unknown service primitive => LLI-Fault.ind <RC = LLI_FMA7_RC9>	**42**	**LLI-READY**
CHECK-CRL LLI CRL check finished \LLI CRL NOT OK OR resources not sufficient => LLI Enable.con(IV/LR) CRL load flag := false	**43**	**LLI-START**
CHECK-CRL LLI CRL check finished \LLI CRL OK AND resources sufficient AND LLI CRL contains at least one communication relationship with a connection attribute unequal ”I" or no connection attribute => predefine dynamic part of the LLI CRL (R)SAP_ACT.req, do not activate LSAP 1 if already activated	**44**	**SAP-ACTIVATE**
CHECK-CRL LLI CRL check finished \valid LLI CRL available AND sufficient resources AND LLI CRL contains only communication relationships with a connection attribute equal ”I” => predefine the dynamic part of the LLI CRL (R)SAP_ACT.req LLI-Enable.con(OK)	**60**	**LLI-READY**
LLI-START LLI CRL check finished \LLI CRL NOT OK OR resources not sufficient	**45**	**LLI-START**
LLI-READY LLI PDU that cannot be assigned to a CREF registered within the LLI CRL => ignore PDU	**46**	**LLI-READY**
LLI-READY any service primitive to the LLI user - LLI interface that cannot be assigned to a Layer 2 address registered within the LLI CRL => ignore service primitive LLI-Fault.ind <RC = LLI_FMA7_RC21>	**47**	**LLI-READY**
SAP-ACTIVATE unallowed FMA1/2 primitive \CRL load flag = false => LLI-Fault.ind<RC = LLI_FMA7_RC22, AD = code of the primitive>	**48**	**ERROR**

Start

Current State Event \Exit Condition => Action Taken	Transition	Next State
SAP-ACTIVATE unallowed FMA1/2 primitive \CRL load flag = true => LLI-Fault.ind<RC = LLI_FMA7_RC22, AD = code of the primitive> LLI Enable.con(IV)	**56**	**ERROR**
SAP-DEACTIVATE unallowed FMA1/2 primitive \SAP_DEACT.req called for all SAPs in {0, 2 to 63, NIL} => LLI-Fault.ind<RC = LLI_FMA7_RC22, AD = code of the primitive> LLI Disable.con	**49**	**LLI-DISABLE**
SAP-DEACTIVATE unallowed FMA1/2 primitive \SAP_DEACT.req not yet called for all SAPs in {0, 2 to 63, NIL} => LLI-Fault.ind<RC = LLI_FMA7_RC22, AD = code of the primitive> SAP_DEACT.req	**50**	**SAP-DEACTIVATE**
SAP-DEACTIVATE SAP_DEACT.con(NO/IV) \SAP_DEACT.req called for all SAPs in {0, 2 to 63, NIL} => LLI-Fault.ind<RC = LLI_FMA7_RC2, AD = M_status> LLI Disable.con	**51**	**LLI-DISABLE**
LOAD-POLL-LIST unallowed FDL primitive received \CRL load flag = false => LLI-Fault.ind<RC = LLI_FMA7_RC23, AD = code of the primitive>	**52**	**ERROR**
LOAD-POLL-LIST unallowed FDL primitive received \CRL load flag = true => LLI-Fault.ind<RC = LLI_FMA7_RC23, AD = code of the primitive> LLI Enable.con(IV)	**58**	**ERROR**

6.7.2 Connection Establishment and Release

If a CREF is reset, the following actions shall be performed:

- Reset STimer, RTimer, T1, T2, T3
- Clear memory
- DTA-Run and T2-Already-Expired shall be set to FALSE
- Reset DTU, DTA, DTC, DTCC and IDLE state machines
- Mark the connection in the CRL as released (STATUS := "CLOSED").
- Predefine the dynamic part of the CRL with the values as defined in section start of LLI.

Upon entry into the state machine for the connection release, the DTU, DTA, DTC acyclic and DTC cyclic state machines shall be stopped. This action is designated in the following formal description as "Stop Machines".

6.7.2.1 State Diagram for Connection Establishment at the Requester

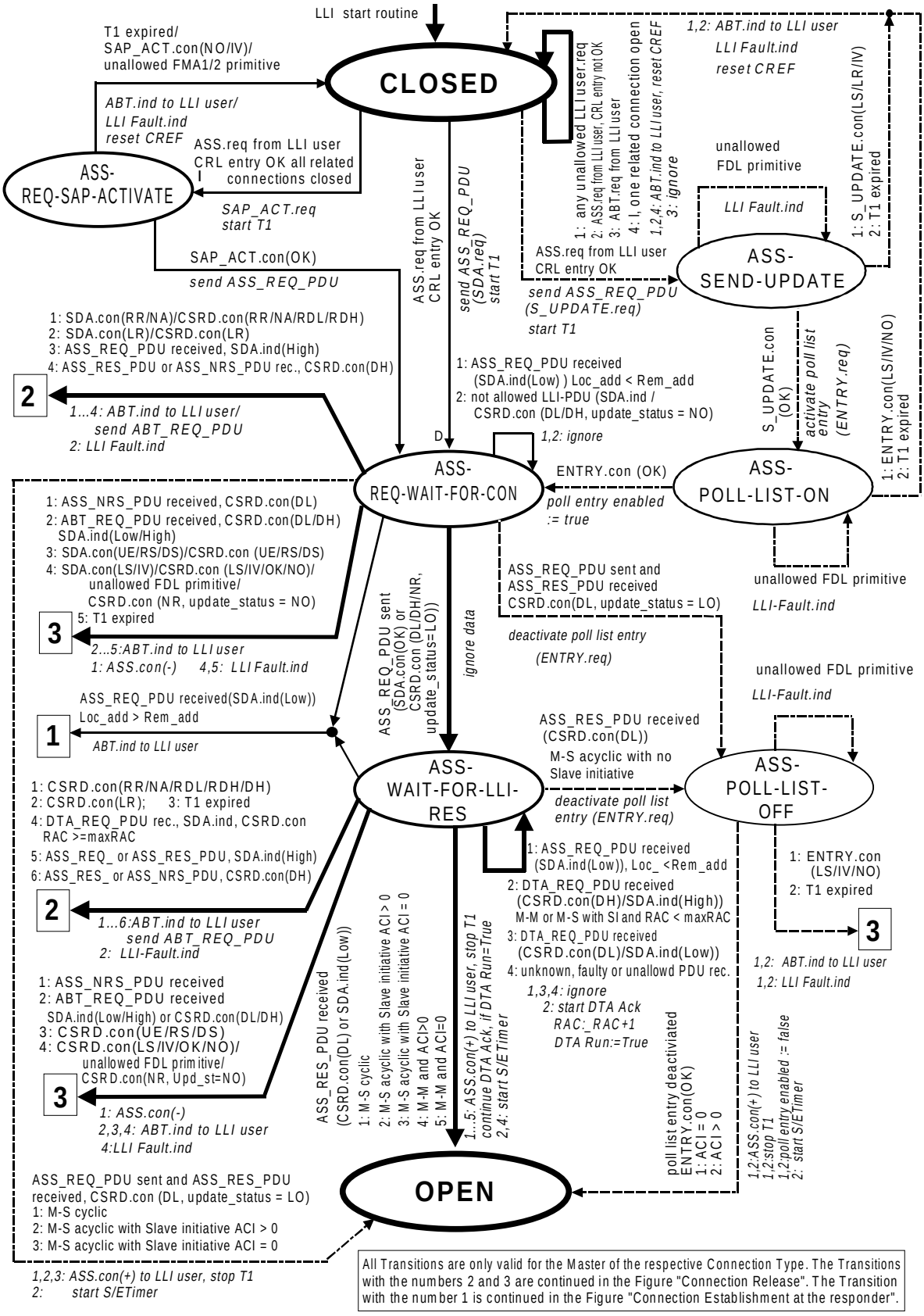


Figure 109. State Diagram for Connection Establishment at the Requester

Description of State Transitions for Connection Establishment at the Requester

All state transitions are valid only for the master of the respective connection type.

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
CLOSED ASS.req from LLI user \(CRL entry OK) AND M-M AND D => send ASS_REQ_PDU (SDA.req(low)), start T1	**ASSREQ 1**	**ASS-REQ-WAIT-FOR-CON**
CLOSED ASS.req from LLI user \(CRL entry OK) AND (M-M AND I) AND all other CREF related machines with identical LSAP in state closed => SAP_ACT.req, start T1 reset CREF	**ASSREQ 2**	**ASS-REQ-SAP-ACTIVATE**
CLOSED ASS.req from LLI user \(CRL entry OK) AND (M-M AND I) AND at least one CREF related machine with identical LSAP not in state closed => ABT.ind to LLI user	**AB 52**	**CLOSED**
CLOSED any unallowed primitive from LLI user \M-M OR M-S => ABT.ind to LLI user <RC = ABT_RC18, AD = code of the primitive> reset CREF	**AB 1**	**CLOSED**
CLOSED ASS.req from LLI user \(CRL entry NOT OK) AND (M-M OR M-S) => ABT.ind to LLI user <RC = ABT_RC20> reset CREF	**AB 2**	**CLOSED**
CLOSED ABT.req from LLI user => ignore	**AB 48**	**CLOSED**
CLOSED ASS.req from LLI user \(CRL entry OK) AND M-S => send ASS_REQ_PDU (S_UPDATE.req), start T1	**ASSREQ 3**	**ASS-SEND-UPDATE**
ASS-REQ-SAP-ACTIVATE SAP_ACT.con(NO/IV) \M-M => ABT.ind to LLI user <RC = ABT_RC13, AD = error state> LLI-Fault.ind <RC = LLI_FMA7_RC1, AD = error state> reset CREF	**AB 3**	**CLOSED**
ASS-REQ-SAP-ACTIVATE SAP_ACT.con(OK) \M-M => send ASS_REQ_PDU (SDA.req(low))	**ASSREQ 5**	**ASS-REQ-WAIT-FOR-CON**

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
ASS-SEND-UPDATE error in loading the update buffer (S_UPDATE.con(LS/LR/IV)) \M-S => ABT.ind to LLI user <RC = LS/LR/IV, AD = ABT_AD1> LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LS/LR/IV> reset CREF	**AB 4**	**CLOSED**
ASS-SEND-UPDATE T1 expired \M-S => ABT.ind to LLI user <RC = ABT_RC10> LLI-Fault.ind <RC = LLI_FMA7_RC18> reset CREF	**AB 5**	**CLOSED**
ASS-SEND-UPDATE unallowed FDL primitive \M-S => LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of prim.>	**ASSREQ 6**	**ASS-SEND-UPDATE**
ASS-SEND-UPDATE update buffer loaded (S_UPDATE.con(OK)) \M-S => activate Poll List entry (ENTRY.req(remote address/remote LSAP/unlock))	**ASSREQ 7**	**ASS-POLL-LIST-ON**
ASS-POLL-LIST-ON error in the activation of the Poll List entry (ENTRY.con(LS/IV/NO)) \M-S => ABT.ind to LLI user <RC = LS/IV/NO, AD = ABT_AD2> LLI-Fault.ind <RC = LLI_FMA7_RC4, AD = LS/IV/NO> reset CREF	**AB 6**	**CLOSED**
ASS-POLL-LIST-ON T1 expired \M-S => ABT.ind to LLI user <RC = ABT_RC10> LLI-Fault.ind <RC = LLI_FMA7_RC18> reset CREF	**AB 7**	**CLOSED**
ASS-POLL-LIST-ON unallowed FDL primitive \M-S => LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = Code of primitive>	**ASSREQ 8**	**ASS-POLL-LIST-ON**
ASS-POLL-LIST-ON Poll List entry activated (ENTRY.con(OK)) \M-S => poll entry enabled := true	**ASSREQ 9**	**ASS-REQ-WAIT-FOR-CON**

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
ASS-REQ-WAIT-FOR-CON unallowed, unknown or faulty LLI PDU received (SDA.ind(serv_class = low/high)) \M-M => ignore	**ASSREQ 10**	**ASS-REQ-WAIT-FOR-CON**
ASS-REQ-WAIT-FOR-CON ASS_REQ_PDU sent (SDA.con(OK)) \M-M	**ASSREQ 11**	**ASS-WAIT-FOR-LLI-RES**
ASS-REQ-WAIT-FOR-CON ASS_REQ_PDU sent (CSRD.con(L_status = DL/DH/NR, update_status = LO)) AND NOT (ASS_RES_PDU OR ASS_NRS_PDU OR ABT_REQ_PDU)received (CSRD.con(L_status = DL, update_status = LO)) \M-S => ignore data if present	**ASSREQ 29**	**ASS-WAIT-FOR-LLI-RES**
ASS-REQ-WAIT-FOR-CON ASS_REQ_PDU sent AND ASS_RES_PDU received (CSRD.con(L_status = DL, update_status = LO)) \cyc. M-S => ASS.con(R+) to FMS stop T1	**ASSREQ 25**	**OPEN**
ASS-REQ-WAIT-FOR-CON ASS_REQ_PDU sent AND ASS_RES_PDU received (CSRD.con(L_status = DL, update_status = LO)) \acyc. M-S with SI AND (ACI > 0) => ASS.con(R+) to FMS stop T1 start S/E Timer	**ASSREQ 26**	**OPEN**
ASS-REQ-WAIT-FOR-CON ASS_REQ_PDU sent AND ASS_RES_PDU received (CSRD.con(L_status = DL, update_status = LO)) \acyc. M-S with SI AND (ACI = 0) => ASS.con(R+) to FMS stop T1	**ASSREQ 27**	**OPEN**
ASS-REQ-WAIT-FOR-CON ASS_REQ_PDU sent AND ASS_RES_PDU received (CSRD.con(L_status = DL, update_status = LO)) \acyc. M-S without SI => deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock))	**ASSREQ 28**	**ASS-POLL-LIST-OFF**
ASS-REQ-WAIT-FOR-CON error on sending the ASS_REQ_PDU (SDA.con(RR/NA)) \M-M => ABT.ind to LLI user <RC = RR/NA, AD = ABT_AD2> send ABT_REQ_PDU <RC = RR/NA> (SDA.req(low)) stop all timers, start T2	**AB 8**	**ABT-SEND-PDU**[1)]

1) see state diagram for connection release

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
ASS-REQ-WAIT-FOR-CON error on sending the ASS_REQ_PDU (SDA.con(LR)) \M-M => ABT.ind to LLI user <RC = LR, AD = ABT_AD4> send ABT_REQ_PDU <RC = LR> (SDA.req(low)) LLI-Fault.ind <RC = LLI_FMA7_RC12, AD = LR> stop all timers, start T2	**AB 9**	**ABT-SEND-PDU**[1)]
ASS-REQ-WAIT-FOR-CON error on sending the ASS_REQ_PDU (CSRD.con(L_status = RR/NA/RDL/RDH)) \M-S => ABT.ind to LLI user <RC = RR/NA/RDL/RDH, AD = ABT_AD5> send ABT_REQ_PDU <RC = RR/NA/RDL/RDH>(S_UPDATE.req) stop all timers, start T2 ignore data if present	**AB 10**	**ABT-UPDATE**[1)]
ASS-REQ-WAIT-FOR-CON error on sending the ASS_REQ_PDU (CSRD.con(L_status = LR)) \M-S => ABT.ind to LLI user <RC = LR, AD = ABT_AD5> send ABT_REQ_PDU <RC = LR> (S_UPDATE.req) LLI-Fault.ind <RC = LLI_FMA7_RC13, AD = LR> stop all timers, start T2	**AB 11**	**ABT-UPDATE**[1)]
ASS-REQ-WAIT-FOR-CON error on sending the ASS_REQ_PDU (SDA.con(UE/RS/DS)) \M-M AND D => ABT.ind to LLI user <RC = UE/RS/DS, AD = ABT_AD4> reset CREF	**AB 12**	**CLOSED**
ASS-REQ-WAIT-FOR-CON T1 expired \M-M AND D => ABT.ind to LLI user <RC = ABT_RC10> reset CREF	**AB 13**	**CLOSED**
ASS-REQ-WAIT-FOR-CON error on sending the ASS_REQ_PDU (SDA.con(UE/RS/DS)) \M-M AND I => ABT.ind to LLI user <RC = UE/RS/DS, AD = ABT_AD4> stop all timers, start T2, SAP_DEACT.req	**AB 14**	**ABT-SAP-DEACTIVATE**[1)]
ASS-REQ-WAIT-FOR-CON T1 expired \M-M AND I => ABT.ind to LLI user <RC = ABT_RC10> LLI-Fault.ind <RC = LLI_FMA7_RC18> stop all timers, start T2, SAP_DEACT.req	**AB 15**	**ABT-SAP-DEACTIVATE**[1)]

1) see state diagram for connection release

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
ASS-REQ-WAIT-FOR-CON error on sending the ASS_REQ_PDU (SDA.con(LS/IV)) \M-M AND D => ABT.ind to LLI user <RC = LS/IV, AD = ABT_AD4> LLI-Fault.ind <RC = LLI_FMA7_RC12, AD = LS/IV> reset CREF	**AB 16**	**CLOSED**
ASS-REQ-WAIT-FOR-CON unallowed FDL primitive \M-M AND D => ABT.ind to LLI user <RC = ABT_RC14, AD = code of the primitive> LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> reset CREF	**AB 17**	**CLOSED**
ASS-REQ-WAIT-FOR-CON error on sending the ASS_REQ_PDU (SDA.con(LS/IV)) \M-M AND I => ABT.ind to LLI user <RC = LS/IV, AD = ABT_AD4> LLI-Fault.ind <RC = LLI_FMA7_RC12, AD = LS/IV> stop all timers, start T2 SAP_DEACT.req	**AB 18**	**ABT-SAP-DEACTIVATE**[1)]
ASS-REQ-WAIT-FOR-CON unallowed FDL primitive \M-M AND I => ABT.ind to LLI user <RC = ABT_RC14, AD = code of the primitive> LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> stop all timers, start T2 SAP_DEACT.req	**AB 19**	**ABT-SAP-DEACTIVATE**[1)]
ASS-REQ-WAIT-FOR-CON error on sending the ASS_REQ_PDU (CSRD.con(L_status = UE/RS/DS)) \M-S => ABT.ind to LLI user <RC = UE/RS/DS, AD = ABT_AD5> deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) stop all timers, start T2	**AB 20**	**ABT-POLL-LIST-OFF**[1)]
ASS-REQ-WAIT-FOR-CON T1 expired \M-S => ABT.ind to LLI user <RC = ABT_RC10> LLI-Fault.ind <RC = LLI_FMA7_RC18> deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) stop all timers, start T2	**AB 21**	**ABT-POLL-LIST-OFF**[1)]

1) see state diagram for connection release

Connection Establishment at the Requester

Current State	Transition	Next State
Event \Exit Condition => Action Taken		

ASS-REQ-WAIT-FOR-CON **AB 22** **ABT-POLL-LIST-OFF**[1)]

error on sending the ASS_REQ_PDU
(CSRD.con(L_status = LS/IV/OK/NO))
\M-S
=> ABT.ind to LLI user <RC = LS/IV/OK/NO, AD = ABT_AD5>
LLI-Fault.ind <RC = LLI_FMA7_RC13, AD = LS/IV/OK/NO>
deactivate Poll List entry
(ENTRY.req(remote address/remote LSAP/lock))
stop all timers, start T2

ASS-REQ-WAIT-FOR-CON **AB 23** **ABT-POLL-LIST-OFF**[1)]

unallowed FDL primitive
\M-S
=> ABT.ind to LLI user <RC=ABT_RC14, AD=code of prim.>
LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive>
deactivate Poll List entry
(ENTRY.req(remote address/remote LSAP/lock))
stop all timers, start T2

ASS-REQ-WAIT-FOR-CON **AB 46** **ABT-POLL-LIST-OFF**[1)]

ASS_REQ_PDU sent AND ASS_NRS_PDU received
(CSRD.con(L_status = DL, update_status = LO))
\M-S
=> ASS.con(R-) to LLI user
stop all timers
start T2
deactivate Poll List entry
(ENTRY.req(remote address/remote LSAP/lock))

ASS-REQ-WAIT-FOR-CON **AB 47** **ABT-POLL-LIST-OFF**[1)]

ASS_REQ_PDU sent AND ABT_REQ_PDU received
(CSRD.con(L_status = DL/DH, update_status = LO))
\M-S
=> ABT.ind to LLI user <RC = RC in ABT_REQ_PDU>
stop all timers, start T2
deactivate Poll List entry
(ENTRY.req(remote address/remote LSAP/lock))

ASS-REQ-WAIT-FOR-CON **ASSREQ 30** **ASS-REQ-WAIT-FOR-CON**

ASS_REQ_PDU received (SDA.ind(serv_class = low))
\M-M AND (local address < remote address)
=> ignore

ASS-REQ-WAIT-FOR-CON **ASSREQ 31** **ASS-WAIT-LOC-RES**[2)]

ASS_REQ_PDU received (SDA.ind(serv_class = low))
\M-M AND (local address > remote address)
AND (LLI-LLI context test OK)
=> ABT.ind to LLI user <RC = ABT_RC21>
ASS.ind to LLI user
start T1

1) see state diagram for connection release
2) see connection establishment at the responder

Connection Establishment at the Requester

Current State	Transition	Next State
Event \Exit Condition => Action Taken		
ASS-REQ-WAIT-FOR-CON	**AB 49**	**ABT-SEND-PDU[1)]**
ASS_REQ_PDU received (SDA.ind(serv_class = low)) \M-M AND (local address > remote address) AND (LLI-LLI context test negative) => ABT.ind to LLI user <RC = ABT_RC21> send ABT_REQ_PDU <RC = ABT_RC1, AD = local LLI context> (SDA.req(low)) stop all timers, start T2		
ASS-REQ-WAIT-FOR-CON	**AB 50**	**ABT-SEND-PDU[1)]**
ASS_REQ_PDU OR ASS_RES_PDU OR ASS_NRS_PDU received (SDA.ind(serv_class = high)) \M-M => ABT.ind to LLI user <RC = ABT_RC8> send ABT_REQ_PDU <RC = ABT_RC8> (SDA.req(low)) stop all timers, start T2		
ASS-REQ-WAIT-FOR-CON	**AB 51**	**ABT-UPDATE[1)]**
ASS_REQ_PDU sent AND (ASS_RES_PDU OR ASS_NRS_PDU received) (CSRA.con(L_status = DH, update_status = LO)) \M-S => ABT.ind to LLI user <RC = ABT_RC8> send ABT_REQ_PDU <RC = ABT_RC8> stop all timers, start T2		
ASS-REQ-WAIT-FOR-CON	**AB 54**	**CLOSED**
ABT_REQ_PDU received (SDA.ind(serv_class = low/high)) \M-M AND D => ABT.ind to LLI user <RC = RC in ABT_REQ_PDU> reset CREF		
ASS-REQ-WAIT-FOR-CON	**AB 53**	**ABT-SAP-DEACTIVATE**
ABT_REQ_PDU received (SDA.ind(serv_class = low/high)) \M-M AND I => ABT.ind to LLI user <RC = RC in ABT_REQ_PDU> stop all timers, start T2 SAP_DEACT.req		
ASS-WAIT-FOR-LLI-RES	**ASSREQ 12**	**OPEN**
ASS_RES_PDU received (SDA.ind(serv_class = low)) \M-M AND (ACI > 0) AND (DTA-Run = FALSE) => ASS.con(R+) to LLI user stop T1, start S/E Timer		
ASS-WAIT-FOR-LLI-RES	**ASSREQ 32**	**OPEN**
ASS_RES_PDU received (SDA.ind(serv_class = low)) \M-M AND (ACI > 0) AND (DTA-Run = TRUE) => ASS.con(R+) to LLI user stop T1, start S/E Timer continue DTA-Ack.(all started DTA-Ack.)		

1) see state diagram for connection release

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
ASS-WAIT-FOR-LLI-RES ASS_RES_PDU received (SDA.ind(serv_class = low)) \M-M AND (ACI = 0) AND (DTA-Run = FALSE) => ASS.con(R+) to LLI user stop T1	**ASSREQ 13**	**OPEN**
ASS-WAIT-FOR-LLI-RES ASS_RES_PDU received (SDA.ind(serv_class = low)) \M-M AND (ACI = 0) AND (DTA-Run = TRUE) => ASS.con(R+) to LLI user stop T1 continue DTA-Ack.(all started DTA-Ack.)	**ASSREQ 33**	**OPEN**
ASS-WAIT-FOR-LLI-RES ASS_RES_PDU received (CSRD.con(L_status = DL)) \cyc. M-S AND (DTA-Run = FALSE) => ASS.con(R+) to FMS stop T1	**ASSREQ 14**	**OPEN**
ASS-WAIT-FOR-LLI-RES ASS_RES_PDU received (CSRD.con(L_status = DL)) \cyc. M-S AND (DTA-Run = TRUE) => ASS.con(R+) to FMS stop T1 continue DTA-Ack.(all started DTA-Ack.)	**ASSREQ 34**	**OPEN**
ASS-WAIT-FOR-LLI-RES ASS_RES_PDU received (CSRD.con(L_status = DL)) \acyc. M-S with SI AND (ACI > 0) AND (DTA-Run = FALSE) => ASS.con(R+) to LLI user stop T1, start S/E Timer	**ASSREQ 15**	**OPEN**
ASS-WAIT-FOR-LLI-RES ASS_RES_PDU received (CSRD.con(L_status = DL)) \acyc. M-S with SI AND (ACI > 0) AND (DTA-Run = TRUE) => ASS.con(R+) to LLI user stop T1, start S/E Timer continue DTA-Ack.(all started DTA-Ack.)	**ASSREQ 35**	**OPEN**
ASS-WAIT-FOR-LLI-RES ASS_RES_PDU received (CSRD.con(L_status = DL)) \acyc. M-S with SI AND (ACI = 0) AND (DTA-Run = FALSE) => ASS.con(R+) to LLI user stop T1	**ASSREQ 16**	**OPEN**
ASS-WAIT-FOR-LLI-RES ASS_RES_PDU received (CSRD.con(L_status = DL)) \acyc. M-S with SI AND (ACI = 0) AND (DTA-Run = TRUE) => ASS.con(R+) to LLI user stop T1 continue DTA-Ack.(all started DTA-Ack.)	**ASSREQ 36**	**OPEN**

Connection Establishment at the Requester

Current State	Transition	Next State
Event \Exit Condition => Action Taken		
ASS-WAIT-FOR-LLI-RES	**ASSREQ 17**	**ASS-WAIT-FOR-LLI-RES**
ASS_REQ_PDU received (SDA.ind(serv_class = low)) \M-M AND (local address < remote address) => ignore		
ASS-WAIT-FOR-LLI-RES	**ASSDTA 1**	**ASS-WAIT-FOR-LLI-RES**
DTA_REQ_PDU received (SDA.ind(serv_class = high)/ CSRD.con(L_status = DH)) \(M-M OR M-S with SI) AND (RAC < max RAC) => start DTA Ack., RAC = RAC + 1, DTA-Run := TRUE		
ASS-WAIT-FOR-LLI-RES	**ASSDTA 2**	**ASS-WAIT-FOR-LLI-RES**
DTA_REQ_PDU received (SDA.ind(serv_class = low)/ CSRD.con(L_status = DL)) \M-M OR M-S => ignore		
ASS-WAIT-FOR-LLI-RES	**ABDTA 1**	**ABT-SEND-PDU[1)]**
DTA_REQ_PDU received (SDA.ind(serv_class = high)) \M-M AND (RAC ≥ max RAC) => ABT.ind to LLI user <RC = ABT_RC6> send ABT_REQ_PDU <RC = ABT_RC6> (SDA.req(low)) stop all timers, start T2, stop machines		
ASS-WAIT-FOR-LLI-RES	**ABDTA 2**	**ABT-UPDATE[1)]**
DTA_REQ_PDU received (CSRD.con(L_status = DH)) \(M-S with SI) AND (RAC ≥ max RAC) => ABT.ind to LLI user <RC = ABT_RC6> send ABT_REQ_PDU <RC = ABT_RC6> (S_UPDATE.req) stop all timers, start T2, stop machines		
ASS-WAIT-FOR-LLI-RES	**ABDTA 3**	**ABT-UPDATE[1)]**
DTA_REQ_PDU received (CSRD.con(L_status = DH)) \M-S without SI => ABT.ind to LLI user <RC = ABT_RC3> send ABT_REQ_PDU <RC = ABT_RC3> (S_UPDATE.req) stop all timers, start T2, stop machines		
ASS-WAIT-FOR-LLI-RES	**ASSREQ 18**	**ASS-WAIT-FOR-LLI-RES**
unknown, faulty or unallowed LLI PDU received (SDA.ind(serv_class = low/high) / CSRD.con (L_status = DL/DH, update_status = NO)) \M-M OR M-S => ignore		
ASS-WAIT-FOR-LLI-RES	**AB 24**	**ABT-SEND-PDU[1)]**
T1 expired \M-M => ABT.ind to LLI user <RC = ABT_RC10> send ABT_REQ_PDU <RC = ABT_RC10> (SDA.req(low)) stop all timers, start T2, stop machines		

1) see state diagram for connection release

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
ASS-WAIT-FOR-LLI-RES error on receipt (CSRD.con(L_status = RR/NA/RDL/RDH/DH)) \M-S => ABT.ind to LLI user <RC = RR/NA/RDL/RDH/DH, AD = ABT_AD7> send ABT_REQ_PDU <RC = RR/NA/RDL/RDH/DH> (S_UPDATE.req) stop all timers, start T2 ignore data if present, stop machines	**AB 25**	**ABT-UPDATE**[1]
ASS-WAIT-FOR-LLI-RES T1 expired \M-S => ABT.ind to LLI user <RC = ABT_RC10> send ABT_REQ_PDU <RC = ABT_RC10> (S_UPDATE.req) stop all timers, start T2, stop machines	**AB 26**	**ABT-UPDATE**[1]
ASS-WAIT-FOR-LLI-RES error on receipt (CSRD.con(L_status = LR)) \M-S => ABT.ind to LLI user <RC = LR, AD = ABT_AD7> LLI-Fault.ind <RC = LLI_FMA7_RC16, AD = LR> send ABT_REQ_PDU <RC = LR> (S_UPDATE.req) stop all timers, start T2, stop machines	**AB 27**	**ABT-UPDATE**[1]
ASS-WAIT-FOR-LLI-RES ASS_REQ_PDU received (SDA.ind(serv_class = low)) \M-M AND (local address > remote address) AND (LLI-LLI context test OK) => ABT.ind to LLI user <RC = ABT_RC21> ASS.ind to LLI user start T1, stop machines	**ASSREQ 20**	**ASS-WAIT-LOC-RES**[2]
ASS-WAIT-FOR-LLI-RES ASS_REQ_PDU received (SDA.ind(serv_class = low)) \M-M AND (local address > remote address) AND (LLI-LLI context test negative) => ABT.ind to LLI user <RC = ABT_RC21> send ABT_REQ_PDU <RC = ABT_RC1, AD = local LLI context> (SDA.req(low)) stop all timers, start T2, stop machines	**AB 28**	**ABT-SEND-PDU**[1]
ASS-WAIT-FOR-LLI-RES ASS_NRS_PDU received (SDA.ind(serv_class = low)) \M-M AND D => ASS.con(R-) to LLI user reset CREF	**AB 29**	**CLOSED**

1) see state diagram for connection release
2) see connection establishment at the responder

Connection Establishment at the Requester

Current State / Event / \Exit Condition / => Action Taken	Transition	Next State
ASS-WAIT-FOR-LLI-RES ASS_NRS_PDU received (SDA.ind(serv_class = low)) \M-M AND I => ASS.con(R-) to LLI user stop all timers start T2 SAP_DEACT.req, stop machines	**AB 30**	**ABT-SAP-DEACTIVATE**[1]
ASS-WAIT-FOR-LLI-RES ABT_REQ_PDU received (SDA.ind(serv_class = low/high)) \M-M AND D => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> reset CREF	**AB 31**	**CLOSED**
ASS-WAIT-FOR-LLI-RES ABT_REQ_PDU received (SDA.ind(serv_class = low/high)) \M-M AND I => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> stop all timers start T2 SAP_DEACT.req, stop machines	**AB 32**	**ABT-SAP-DEACTIVATE**[1]
ASS-WAIT-FOR-LLI-RES ASS_NRS_PDU received (CSRD.con(L_status = DL)) \M-S => ASS.con(R-) to LLI user stop all timers, start T2, stop machines deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock))	**AB 33**	**ABT-POLL-LIST-OFF**[1]
ASS-WAIT-FOR-LLI-RES ASS_NRS_PDU or ASS_RES_PDU received (CSRD.con(L_status = DH)) \M-S => ABT.ind to LLI user <RC = ABT_RC8> send ABT_REQ_PDU <RC = ABT_RC8> (S_UPDATE.req) stop all timers, start T2, stop machines	**AB 55**	**ABT-UPDATE**[1]
ASS-WAIT-FOR-LLI-RES ABT_REQ_PDU received (CSRD.con(L_status = DL/DH)) \M-S => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> stop all timers, start T2, stop machines deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock))	**AB 34**	**ABT-POLL-LIST-OFF**[1]
ASS-WAIT-FOR-LLI-RES error on receipt ((CSRD.con(L_status = UE/RS/DS)) \M-S => ABT.ind to LLI user <RC = UE/RES/DS, AD = ABT_AD7> stop all timers, start T2, stop machines deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock))	**AB 35**	**ABT-POLL-LIST-OFF**[1]

1) see state diagram for connection release

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
ASS-WAIT-FOR-LLI-RES error on receipt (CSRD.con(L_status = LS/IV/OK/NO)) \M-S => ABT.ind to LLI user <RC = LS/IV/OK/NO, AD = ABT_AD7> LLI-Fault.ind <RC = LLI_FMA7_RC16, AD = LS/IV/OK/NO> stop all timers, start T2, stop machines deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock))	**AB 36**	**ABT-POLL-LIST-OFF**[1)]
ASS-WAIT-FOR-LLI-RES ASS_REQ_PDU OR ASS_RES_PDU OR ASS_NRS_PDU received (SDA.ind(serv_class = high)) \M-M => ABT.ind to LLI user <RC = ABT_RC8> send ABT_REQ_PDU <RC = ABT_RC8> (SDA.req(low)) stop all timers, start T2, stop machines	**AB 56**	**ABT-SEND-PDU**[1)]
ASS-WAIT-FOR-LLI-RES ASS_RES_PDU received (CSRD.con(L_status = DL)) \acyc. M-S without SI => deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) stop machines	**ASSREQ 21**	**ASS-POLL-LIST-OFF**
ASS-POLL-LIST-OFF Poll List entry deactivated (ENTRY.con(OK)) \M-S AND (ACI > 0) => ASS.con(R+) to LLI user poll entry enabled := false stop T1, start S/RTimer	**ASSREQ 22**	**OPEN**
ASS-POLL-LIST-OFF Poll List entry deactivated (ENTRY.con(OK)) \M-S AND (ACI = 0) => ASS.con(R+) to LLI user poll entry enabled := false, stop T1	**ASSREQ 23**	**OPEN**
ASS-POLL-LIST-OFF unallowed FDL primitive \M-S => LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive>	**ASSREQ 24**	**ASS-POLL-LIST-OFF**
ASS-POLL-LIST-OFF error in the deactivation of the Poll List entry (ENTRY.con(LS/IV/NO)) \M-S => ABT.ind to LLI user <RC = LS/IV/NO, AD = ABT_AD3> LLI-Fault.ind <RC = LLI_FMA7_RC5, AD = LS/IV/NO> stop all timers, start T2 deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock))	**AB 37**	**ABT-POLL-LIST-OFF**[1)]

1) see state diagram for connection release

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
ASS-POLL-LIST-OFF T1 expired \M-S => ABT.ind to LLI user <RC = ABT_RC10> LLI-Fault.ind <RC = LLI_FMA7_RC18> stop all timers, start T2 deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock))	**AB 38**	**ABT-POLL-LIST-OFF**[1)]
ASS-REQ-SAP-ACTIVATE T1 expired \M-M => ABT.ind to LLI user <RC = ABT_RC10> LLI-Fault.ind <RC = LLI_FMA7_RC18> reset CREF	**AB 39**	**CLOSED**
ASS-WAIT-FOR-LLI-RES unexpected FDL primitive \M-S => ABT.ind to LLI user <RC = ABT_RC14, AD = code of the primitive> LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> stop all timers, start T2, stop machines deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock))	**AB 40**	**ABT-POLL-LIST-OFF**[1)]
ASS-WAIT-FOR-LLI-RES unexpected FDL primitive \M-M AND D => ABT.ind to LLI user <RC = ABT_RC14, AD = code of the primitive> LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> stop all timers, stop machines	**AB 41**	**CLOSED**
ASS-WAIT-FOR-LLI-RES unexpected FDL primitive \M-M AND I => ABT.ind to LLI user <RC = ABT_RC14, AD = code of the primitive> LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> stop all timers, start T2, stop machines SAP_DEACT.req	**AB 42**	**ABT-SAP-DEACTIVATE**[1)]
ASS-REQ-WAIT-FOR-CON error on sending the ASS_REQ_PDU (CSRD.con(L_status = NR, update_status = NO)) \M-S => ABT.ind to LLI user <RC = ABT_RC25> LLI-Fault.ind <RC = LLI_FMA7_RC24, AD = NR> deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) stop all timers, start T2	**AB 43**	**ABT-POLL-LIST-OFF**[1)]

1) see state diagram for connection release

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
ASS-WAIT-FOR-LLI-RES error on receipt CSRD.con(L_status = NR, update_status = NO)) \M-S => ABT.ind to LLI user <RC = ABT_RC25> LLI-Fault.ind <RC = LLI_FMA7_RC24, AD = NR> stop all timers, start T2, stop machines deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock))	**AB 44**	**ABT-POLL-LIST-OFF**[1)]
ASS-REQ-SAP-ACTIVATE unallowed FMA1/2 primitive \M-M => ABT.ind to LLI user <RC = ABT_RC26> LLI-Fault.ind <RC = LLI_FMA7_RC22> reset CREF	**AB 45**	**CLOSED**

1) see state diagram for connection release

6.7.2.2 State Diagram for Connection Establishment at the Responder

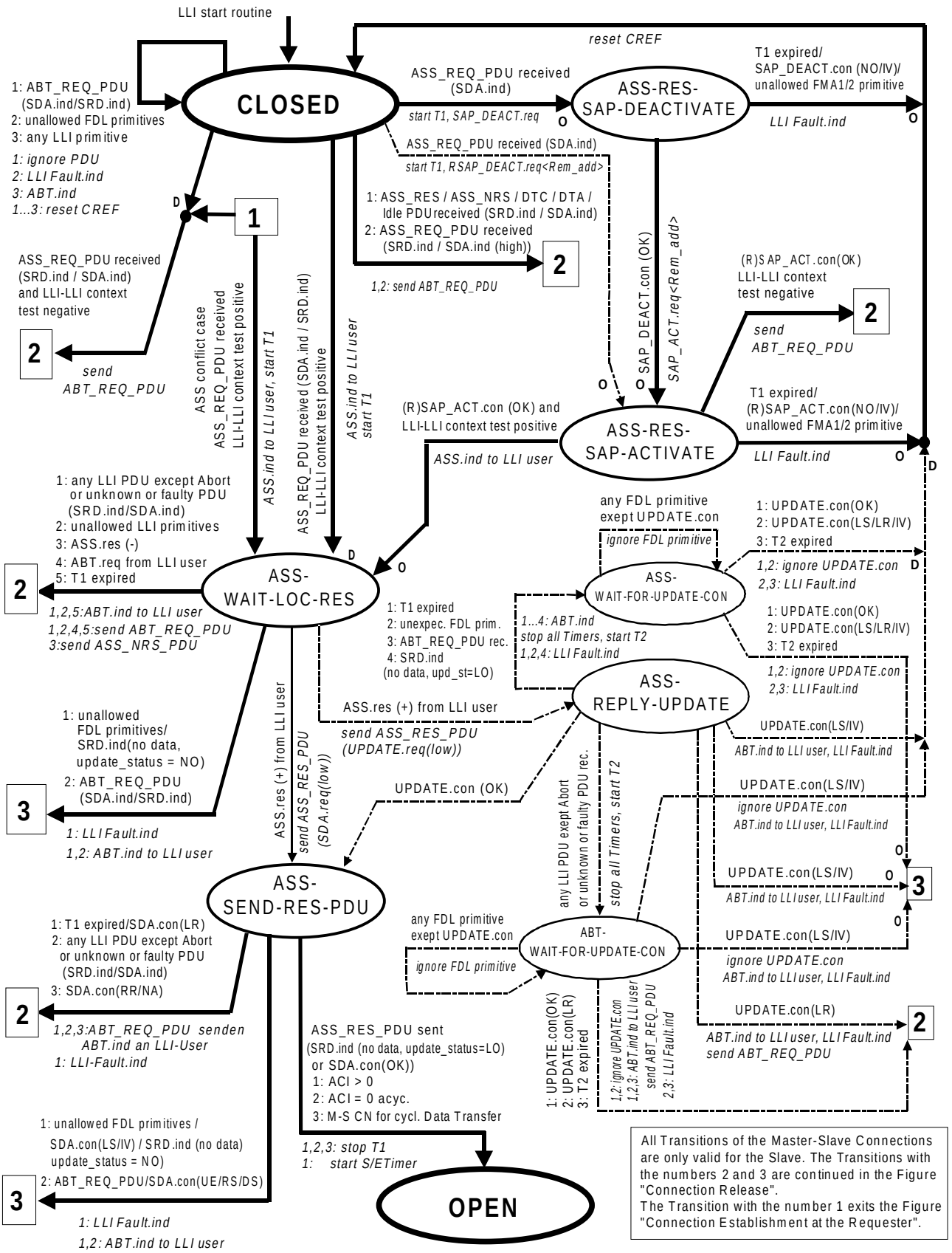


Figure 110. State Diagram for Connection Establishment at the Responder

Description of State Transitions for Connection Establishment at the Responder

All state transitions of the master-slave connections are only valid for the slave.

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
CLOSED ABT_REQ_PDU received (SDA.ind(serv_class = low/high)/ SRD.ind(serv_class = low/high)) \M-M OR M-S => ignore PDU reset CREF	**AB 1**	**CLOSED**
CLOSED unallowed FDL primitive \M-M OR M-S => LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> reset CREF	**AB 2**	**CLOSED**
CLOSED ASS_REQ_PDU received (SDA.ind(serv_class = low)) \M-M AND (LLI-LLI context test negative) AND D => send ABT_REQ_PDU <RC = ABT_RC1, AD = local LLI context> stop all timers, start T2	**AB 3**	**ABT-SEND-PDU**[1)]
CLOSED ASS_REQ_PDU received (SRD.ind(serv_class = low)) \M-S AND (LLI-LLI context test negative) AND D => send ABT_REQ_PDU <RC=ABT_RC1, AD=loc. LLI context> (UPDATE.req(low)) stop all timers, start T2	**AB 4**	**ABT-UPDATE**[1)]
CLOSED ASS_RES_PDU received (SDA.ind(serv_class = low/high)) OR ASS_NRS_PDU received (SDA.ind(serv_class = low/high)) OR DTC_REQ_PDU received (SDA.ind(serv_class = low/high)) OR DTC_RES_PDU received (SDA.ind(serv_class = low/high)) OR DTA_REQ_PDU received (SDA.ind(serv_class = low/high)) OR DTA_ACK_PDU received (SDA.ind(serv_class = low/high)) OR IDLE_REQ_PDU received (SDA.ind(serv_class = low/high) \M-M AND D => send ABT_REQ_PDU <RC = ABT_RC2> (SDA.req(low)) stop all timers, start T2	**AB 5**	**ABT-SEND-PDU**[1)]
CLOSED ASS_REQ_PDU received (SDA.ind(serv_class = high)) \M-M AND D => send ABT_REQ_PDU <RC = ABT_RC8> (SDA.req(low)) stop all timers, start T2	**AB 61**	**ABT-SEND-PDU**[1)]

1) see state diagram for connection release

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
CLOSED	**AB 6** ASS_RES_PDU received (SRD.ind(serv_class = low/high)) OR ASS_NRS_PDU received (SRD.ind(serv_class = low/high)) OR DTC_REQ_PDU received (SRD.ind(serv_class = low/high)) OR DTC_RES_PDU received (SRD.ind(serv_class = low/high)) OR DTA_REQ_PDU received (SRD.ind(serv_class = low/high)) OR DTA_ACK_PDU received (SRD.ind(serv_class = low/high)) OR IDLE_REQ_PDU received (SRD.ind(serv_class = low/high)) \M-S => send ABT_REQ_PDU <RC = ABT_RC2> (UPDATE.req(low)) stop all timers, start T2	**ABT-UPDATE**[1)]
CLOSED	**AB 62** ASS_REQ_PDU received (SRD.ind(serv_class = high)) \M-S => send ABT_REQ_PDU <RC = ABT_RC8> (UPDATE.req(low)) stop all timers, start T2	**ABT-UPDATE**[1)]
CLOSED	**AB 63** ASS_RES_PDU received (SDA.ind(serv_class = low/high)) OR ASS_NRS_PDU received (SRD.ind(serv_class = low/high)) OR DTC_REQ_PDU received (SDA.ind(serv_class = low/high)) OR DTC_RES_PDU received (SDA.ind(serv_class = low/high)) OR DTA_REQ_PDU received (SDA.ind(serv_class = low/high)) OR DTA_ACK_PDU received (SDA.ind(serv_class = low/high)) OR IDLE_REQ_PDU received (SDA.ind(serv_class = low/high) \M-M AND O => send ABT_REQ_PDU <RC = ABT_RC2> (SDA.req(low)) use Layer 2 address out of the received PDU stop all timers, start T2	**ABT-SEND-PDU**[1)]
CLOSED	**AB 64** ASS_REQ_PDU received (SDA.ind(serv_class = high)) \M-M AND O => send ABT_REQ_PDU <RC = ABT_RC8> (SDA.req(low)) use Layer 2 address out of the received PDU stop all timers, start T2	**ABT-SEND-PDU**[1)]
CLOSED	**AB 69** Any unallowed primitive from LLI user \(M-S AND Slave) OR (MM AND O) => ABT.ind to LLI user <RC = ABT_RC18, AD = Code of the primitive> reset CREF	**CLOSED**
CLOSED	**ASSRES 1** ASS_REQ_PDU received (SDA.ind(serv_class = low)/ SRD.ind(serv_class = low)) \(M-M OR M-S) AND (LLI-LLI context test positive) AND D => ASS.ind to LLI user start T1	**ASS-WAIT-LOC-RES**

1) see state diagram for connection release

Connection Establishment at the Requester

Current State **Transition** **Next State**
Event
\Exit Condition
=> Action Taken

CLOSED **ASSRES 2** **ASS-RES-SAP-DEACTIVATE**
ASS_REQ_PDU received (SDA.ind(serv_class = low))
\M-M AND O
=> SAP_DEACT.req
store FDL_address and FDL_SAP from SRD.ind to actual remote address and actual remote SAP in CRL
store ASS_REQ_PDU
start T1

CLOSED **ASSRES 14** **ASS-RES-SAP-ACTIVATE**
ASS_REQ_PDU received (SRD.ind(serv_class = low))
\M-S AND O
=> RSAP-ACT.req(Access = FDL address from SRD.ind,Indication_mode = unchanged)
store FDL_address and FDL_SAP from SRD.ind to actual remote address and actual remote SAP in CRL
store ASS_REQ_PDU
start T1

ASS-RES-SAP-DEACTIVATE **AB 7** **CLOSED**
SAP_DEACT.con(NO/IV))
\M-M
=> LLI-Fault.ind <RC = LLI_FMA7_RC2, AD = error state>
reset CREF

ASS-RES-SAP-DEACTIVATE **ASSRES 4** **ASS-RES-SAP-ACTIVATE**
SAP_DEACT.con(OK)
\M-M
=> SAP_ACT.req

ASS-RES-SAP-ACTIVATE **AB 8** **CLOSED**
(R)SAP_ACT.con(NO/IV)
\M-M OR M-S
=> LLI-Fault.ind <RC = LLI_FMA7_RC1, AD = error state>
reset CREF

ASS-RES-SAP-ACTIVATE **ASSRES 7** **ASS-WAIT-LOC-RES**
(R)SAP_ACT.con(OK)
\(M-M OR M-S) AND (LLI-LLI context test positive)
=> ASS.ind to LLI user

ASS-RES-SAP-ACTIVATE **AB 9** **ABT-SEND-PDU[1)]**
SAP_ACT.con(OK)
\M-M AND (LLI-LLI context test negative)
=> send ABT_REQ_PDU
<RC = ABT_RC1, AD = local LLI context> (SDA.req(low))
stop all timers, start T2

1) see state diagram for connection release

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
ASS-RES-SAP-ACTIVATE RSAP_ACT.con(OK) \M-S AND (LLI-LLI context test negative) => send ABT_REQ_PDU <RC = ABT_RC1, AD=loc. LLI context> (UPDATE.req(low)) stop all timers, start T2	**AB 10**	**ABT-UPDATE**[1)]
ASS-WAIT-LOC-RES ASS.res(R+) from LLI user \M-S => send ASS_RES_PDU (UPDATE.req(low))	**ASSRES 8**	**ASS-REPLY-UPDATE**
ASS-WAIT-LOC-RES ASS.res(R+) from LLI user \M-M => send ASS_RES_PDU (SDA.req(low))	**ASSRES 9**	**ASS-SEND-RES-PDU**
ASS-WAIT-LOC-RES unallowed LLI primitive \M-M => ABT.ind to LLI user <RC=ABT_RC18, AD=code of prim.> send ABT_REQ_PDU <RC = ABT_RC9> (SDA.req(low)) stop all timers, start T2	**AB 11**	**ABT-SEND-PDU**[1)]
ASS-WAIT-LOC-RES T1 expired \M-M => ABT.ind to LLI user <RC = ABT_RC10> send ABT_REQ_PDU <RC = ABT_RC10> (SDA.req(low)) stop all timers, start T2	**AB 12**	**ABT-SEND-PDU**[1)]
ASS-WAIT-LOC-RES ASS.res(R-) from LLI user \M-M => send ASS_NRS_PDU (SDA.req(low)) stop all timers, start T2	**AB 13**	**ABT-SEND-PDU**[1)]
ASS-WAIT-LOC-RES ABT.req from LLI user \M-M => send ABT_REQ_PDU (SDA.req(low)) stop all timers, start T2	**AB 14**	**ABT-SEND-PDU**[1)]
ASS-WAIT-LOC-RES unallowed LLI primitive \M-S => ABT.ind to LLI user <RC = ABT_RC18, AD = code of the primitive> send ABT_REQ_PDU <RC = ABT_RC9> (UPDATE.req(low)) stop all timers, start T2	**AB 15**	**ABT-UPDATE**[1)]

1) see state diagram for connection release

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
ASS-WAIT-LOC-RES T1 expired \M-S	**AB 16** => ABT.ind to LLI user <RC = ABT_RC10> send ABT_REQ_PDU <RC = ABT_RC10> (UPDATE.req(low)) stop all timers, start T2	**ABT-UPDATE**[1)]
ASS-WAIT-LOC-RES ASS.res(R-) from LLI user \M-S	**AB 17** => send ASS_NRS_PDU (UPDATE.req(low)) stop all timers, start T2	**ABT-UPDATE**[1)]
ASS-WAIT-LOC-RES ABT.req from LLI user \M-S	**AB 18** => send ABT_REQ_PDU (UPDATE.req(low)) stop all timers, start T2	**ABT-UPDATE**[1)]
ASS-SEND-RES-PDU ASS_RES_PDU sent (SDA.con(OK)/ SRD.ind(no data, update_status = LO)) \(M-M OR M-S) acyc. AND (ACI > 0)	**ASSRES 10** => stop T1, start S/RTimer	**OPEN**
ASS-SEND-RES-PDU ASS_RES_PDU sent (SDA.con(OK)/ SRD.ind(no data, update_status = LO)) \(M-M OR M-S)acyc. AND (ACI = 0)	**ASSRES 11** => stop T1	**OPEN**
ASS-SEND-RES-PDU ASS_RES_PDU sent (SRD.ind(no data, update_status = LO)) \cyc. M-S	**ASSRES 12** => stop T1	**OPEN**
ASS-SEND-RES-PDU T1 expired \M-M	**AB 19** => send ABT_REQ_PDU <RC = ABT_RC10> (SDA.req(low)) ABT.ind to LLI user <RC = ABT_RC10> LLI-Fault.ind <RC = LLI_FMA7_RC18> stop all timers, start T2	**ABT-SEND-PDU**[1)]
ASS-SEND-RES-PDU error on sending the ASS_RES_PDU (SDA.con(LR)) \M-M	**AB 20** => send ABT_REQ_PDU <RC = LR> (SDA.req(low)) ABT.ind to LLI user <RC = LR, AD = ABT_AD4> LLI-Fault.ind <RC = LLI_FMA7_RC12, AD = LR> stop all timers, start T2	**ABT-SEND-PDU**[1)]

1) see state diagram for connection release

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
ASS-SEND-RES-PDU T1 expired \M-S => send ABT_REQ_PDU <RC = ABT_RC10> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC10> LLI-Fault.ind <RC = LLI_FMA7_RC18> stop all timers, start T2	**AB 21**	**ABT-UPDATE**[1)]
ASS-SEND-RES-PDU any received LLI PDU except ABT_REQ_PDU (SDA.ind(low/high)) \M-M => send ABT_REQ_PDU <RC = ABT_RC2> (SDA.req(low)) ABT.ind to LLI user <RC = ABT_RC2> stop all timers, start T2	**AB 22**	**ABT-SEND-PDU**[1)]
ASS-SEND-RES-PDU unknown or faulty PDU received (SDA.ind(low/high)) \M-M => send ABT_REQ_PDU <RC = ABT_RC4> (SDA.req(low)) ABT.ind to LLI user <RC = ABT_RC4> stop all timers, start T2	**AB 89**	**ABT-SEND-PDU**[1)]
ASS-SEND-RES-PDU error on sending the ASS_RES_PDU (SDA.con(RR/NA)) \M-M => send ABT_REQ_PDU <RC = RR/NA> (SDA.req(low)) ABT.ind to LLI user <RC = RR/NA, AD = ABT_AD4> stop all timers, start T2	**AB 23**	**ABT-SEND-PDU**[1)]
ASS-SEND-RES-PDU any received LLI PDU except ABT_REQ_PDU (SRD.ind(serv_class = low/high)) \M-S => send ABT_REQ_PDU <RC = ABT_RC2> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC2> stop all timers, start T2	**AB 24**	**ABT-UPDATE**[1)]
ASS-SEND-RES-PDU unknown or faulty PDU received (SRD.ind(serv_class = low/high)) \M-S => send ABT_REQ_PDU <RC = ABT_RC4> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC4> stop all timers, start T2	**AB 90**	**ABT-UPDATE**[1)]
ASS-SEND-RES-PDU unallowed FDL primitive \D => LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> ABT.ind to LLI user <RC = ABT_RC14, AD = code of the primitive> reset CREF	**AB 25**	**CLOSED**

1) see state diagram for connection release

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
ASS-SEND-RES-PDU error on sending the ASS_RES_PDU (SDA.con(LS/IV)) \M-M AND D => LLI-Fault.ind <RC = LLI_FMA7_RC12, AD = LS/IV> ABT.ind to LLI user <RC = LS/IV, AD = ABT_AD4> reset CREF	**AB 26**	**CLOSED**
ASS-SEND-RES-PDU ABT_REQ_PDU received (SDA.ind(serv_class = low/high)) \M-M AND D => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> reset CREF	**AB 27**	**CLOSED**
ASS-WAIT-LOC-RES ABT_REQ_PDU received (SDA.ind(serv_class = low/high)) \M-M AND D => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> reset CREF	**AB 28**	**CLOSED**
ASS-SEND-RES-PDU error on sending the ASS_RES_PDU (SDA.con(UE/RS/DS)) \M-M AND D => ABT.ind to LLI user <RC = UE/RS/DS, AD = ABT_AD4> reset CREF	**AB 29**	**CLOSED**
ASS-SEND-RES-PDU ABT_REQ_PDU received (SRD.ind(serv_class = low/high)) \M-S AND D => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> reset CREF	**AB 30**	**CLOSED**
ASS-SEND-RES-PDU unallowed FDL primitive \M-M AND O => LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> ABT.ind to LLI user <RC = ABT_RC14, AD = code of the primitive> stop all timers, start T2 SAP_DEACT.req	**AB 31**	**ABT-SAP-DEACTIVATE[1)]**
ASS-SEND-RES-PDU error on sending the ASS_RES_PDU (SDA.con(LS/IV)) \M-M AND O => LLI-Fault.ind <RC = LLI_FMA7_RC12, AD = LS/IV> ABT.ind to LLI user <RC = LS/IV, AD = ABT_AD4> stop all timers, start T2 SAP_DEACT.req	AB 32	ABT-SAP-DEACTIVATE[1)]
ASS-SEND-RES-PDU unallowed FDL primitive \M-S AND O => LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> ABT.ind to LLI user <RC = ABT_RC14, AD = code of the primitive> stop all timers, start T2 RSAP-ACTIVATE.req (ACCESS = All, Indication_Mode unchanged)	AB 33	ABT-SAP-ACTIVATE[1)]

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
ASS-SEND-RES-PDU ABT_REQ_PDU received (SDA.ind(serv_class = low/high)) \M-M AND O => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> stop all timers, start T2 SAP_DEACT.req	**AB 34**	**ABT-SAP-DEACTIVATE**[1)]
ASS-SEND-RES-PDU error on sending the ASS_RES_PDU (SDA.con(UE/RS/DS)) \M-M AND O => ABT.ind to LLI user <RC = UE/RS/DS, AD = ABT_AD4> stop all timers, start T2 SAP_DEACT.req	**AB 35**	**ABT-SAP-DEACTIVATE**[1)]
ASS-SEND-RES-PDU ABT_REQ_PDU received (SRD.ind(serv_class = low/high)) \M-S AND O => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> stop all timers, start T2 RSAP-ACTIVATE.req (ACCESS = All, Indication_Mode unchanged)	**AB 36**	**ABT-SAP-ACTIVATE**[1)]
ASS-REPLY-UPDATE error in loading the update buffer (UPDATE.con(LS/IV)) \M-S AND D => ABT.ind to LLI user <RC = LS/IV, AD = ABT_AD1> LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LS/IV> reset CREF	**AB 37**	**CLOSED**
ASS-REPLY-UPDATE T1 expired \M-S => ABT.ind to LLI user <RC = ABT_RC10> LLI-Fault.ind <RC = LLI_FMA7_RC18> stop all timers, start T2	**AB 38**	**ASS-WAIT-FOR-UPDATE-CON**
ASS-REPLY-UPDATE unallowed FDL primitive \M-S => ABT.ind to LLI user <RC = ABT_RC14, AD = code of the primitive> LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> stop all timers, start T2	**AB 39**	**ASS-WAIT-FOR-UPDATE-CON**
ASS-REPLY-UPDATE ABT_REQ_PDU received (SRD.ind(serv_class = low/high)) \M-S => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> stop all timer, start T2	**AB 40**	**ASS-WAIT-FOR-UPDATE-CON**
ASS-REPLY-UPDATE update buffer loaded (UPDATE.con(OK)) \M-S	**ASSRES 13**	**ASS-SEND-RES-PDU**

1) see state diagram for connection release

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
ASS-REPLY-UPDATE error in loading the update buffer (UPDATE.con(LS/IV)) \M-S AND O => ABT.ind to LLI user <RC = LS/IV, AD = ABT_AD1> LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LS/IV> stop all timers, start T2 RSAP-ACTIVATE.req (ACCESS = All, Indication_Mode unchanged)	**AB 41**	**ABT-SAP-ACTIVATE 1)**
ASS-REPLY-UPDATE error in loading the update buffer (UPDATE.con(LR)) \M-S => ABT.ind to LLI user <RC = LR, AD = ABT_AD1> LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LR> send ABT_REQ_PDU <RC = LR> (UPDATE.req(low)) stop all timers, start T2	**AB 45**	**ABT-UPDATE 1)**
ASS-REPLY-UPDATE any received LLI PDU except ABT_REQ_PDU (SRD.ind(low/high)) \M-S => stop all timers, start T2 store RC:=ABT_RC2	**AB 46**	**ABT-WAIT-FOR-UPDATE-CON**
ASS-REPLY-UPDATE unknown or faulty PDU received (SRD.ind(low/high)) \M-S => stop all timers, start T2 store RC:=ABT_RC4	**AB 86**	**ABT-WAIT-FOR-UPDATE-CON**
ASS-RES-SAP-DEACTIVATE T1 expired \M-M => LLI-Fault.ind <RC = LLI_FMA7_RC18> reset CREF	**AB 47**	**CLOSED**
ASS-RES-SAP-ACTIVATE T1 expired \M-M OR M-S => LLI-Fault.ind <RC = LLI_FMA7_RC18> reset CREF	**AB 48**	**CLOSED**
ASS-WAIT-LOC-RES any received LLI PDU except ABT_REQ_PDU (SDA.req(low/high)) \M-M => send ABT_REQ_PDU <RC = ABT_RC2> (SDA.req(low)) ABT.ind to LLI user <RC = ABT_RC2> stop all timers, start T2	**AB 49**	**ABT-SEND-PDU 1)**

1) see state diagram for connection release

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
ASS-WAIT-LOC-RES any received LLI PDU except ABT_REQ_PDU (SRD.ind(serv_class = low/high)) \M-S => send ABT_REQ_PDU <RC = ABT_RC2> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC2> stop all timers, start T2	**AB 50**	**ABT-UPDATE[1)]**
ASS-WAIT-LOC-RES unknown or faulty PDU received (SDA.req(low/high)) \M-M => send ABT_REQ_PDU <RC = ABT_RC4> (SDA.req(low)) ABT.ind to LLI user <RC = ABT_RC4> stop all timers, start T2	**AB 83**	**ABT-SEND-PDU[1)]**
ASS-WAIT-LOC-RES unknown or faulty PDU received (SRD.ind(serv_class = low/high)) \M-S => send ABT_REQ_PDU <RC = ABT_RC4> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC4> stop all timers, start T2	**AB 84**	**ABT-UPDATE[1)]**
ASS-WAIT-LOC-RES unallowed FDL primitive \D => LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> ABT.ind to LLI user <RC = ABT_RC14, AD = code of the primitive> reset CREF	**AB 51**	**CLOSED**
ASS-WAIT-LOC-RES ABT_REQ_PDU received (SRD.ind(serv_class = low/high)) \M-S AND D => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> reset CREF	**AB 52**	**CLOSED**
ASS-WAIT-LOC-RES ABT_REQ_PDU received (SRD.ind(serv_class = low/high)) \M-S AND O => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> stop all timers, start T2 RSAP_ACT.req(Access = All, Indication_mode = Unchanged)	**AB 68**	**ABT-SAP-ACTIVATE**
ASS-WAIT-LOC-RES ABT_REQ_PDU received (SDA.ind(serv_class = low/high)) \M-M AND O => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> stop all timers, start T2 SAP_DEACT.req	**AB 70**	**ABT-SAP-DEACTIVATE**
ASS-WAIT-LOC-RES ABT_REQ_PDU received (SDA.ind(serv_class = low/high)) \M-M AND D => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> reset CREF	AB 85	CLOSED

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
ASS-REPLY-UPDATE SRD.ind(no data, update_status = NO) \M-S => ABT.ind to LLI user <RC = ABT_RC25> LLI-Fault.ind <RC = LLI_FMA7_RC24> stop all timers, start T2	**AB 53**	**ASS-WAIT-FOR-UPDATE-CON**
LLI-Fault **ASS-WAIT-LOC-RES** SRD.ind(no data, update_status = NO) \M-S AND D => ABT.ind to LLI user <RC = ABT_RC25> LLI-Fault.ind <RC = LLI_FMA7_RC24> reset CREF	**AB 55**	**CLOSED**
ASS-WAIT-LOC-RES SRD.ind(no data, update_status = NO) \M-S AND O => ABT.ind to LLI user <RC = ABT_RC25> LLI-Fault.ind <RC = LLI_FMA7_RC24> stop all timers, start T2 RSAP-ACTIVATE.req (ACCESS = All, Indication_Mode unchanged)	**AB 56**	**ABT-SAP-ACTIVATE**[1)]
ASS-WAIT-LOC-RES unallowed FDL-Primitive \M-M AND O =>LLI-Fault.ind <RC = LLI_FMA7_RC6, AD= Code of the primitive> ABT.ind to LLI-User <RC = ABT_RC_14, AD = Code ofe the primitive> stop all timer,start T2 SAP_DEACT.req	**AB 67**	**ABT-SAP-DEACTIVATE**
ASS-WAIT-LOC-RES unallowed FDL-Primitive \M-S AND Slave AND O =>LLI-Fault.ind <RC = LLI_FMA7_RC6, AD= Code of the primitive> ABT.ind to LLI-User <RC = ABT_RC_14, AD = Code of the primitive> stop all timer,start T2 RSAP_ACT.req(Access = All, Indication_mode = Unchanged)	**AB 76**	**ABT-SAP-ACTIVATE**
ASS-SEND-RES-PDU SRD.ind(no data, update_status = NO) \M-S AND D => ABT.ind to LLI user <RC = ABT_RC25> LLI-Fault.ind <RC = LLI_FMA7_RC24> reset CREF	**AB 57**	**CLOSED**

1) see state diagram for connection release

Connection Establishment at the Requester

Current State / Event / \Exit Condition / => Action Taken	Transition	Next State
ASS-SEND-RES-PDU SRD.ind(no data, update_status = NO) \M-S AND O => ABT.ind to LLI user <RC = ABT_RC25> LLI-Fault.ind <RC = LLI_FMA7_RC24> stop all timers, start T2 RSAP-ACTIVATE.req (ACCESS = All, Indication_Mode =unchanged)	**AB 58**	**ABT-SAP-ACTIVATE[1)]**
ASS-RES-SAP-DEACTIVATE unallowed FMA1/2 primitive received \M-M => LLI-Fault.ind <RC = LLI_FMA7_RC22> reset CREF	**AB 59**	**CLOSED**
ASS-RES-SAP-ACTIVATE unallowed FMA1/2 primitive received \M-M OR M-S => LLI-Fault.ind <RC = LLI_FMA7_RC22> reset CREF	**AB 60**	**CLOSED**
ASS-WAIT-FOR-UPDATE-CON UPDATE.CON(OK) \M-S AND D => ignore UPDATE.CON reset CREF	**AB 71**	**CLOSED**
ASS-WAIT-FOR-UPDATE-CON UPDATE.CON(LS/LR/IV) \M-S AND D => ignore UPDATE.CON reset CREF LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LS/LR/IV)	**AB 82**	**CLOSED**
ASS-WAIT-FOR-UPDATE-CON T2 expired \M-S AND D => LLI-Fault.ind <RC = LLI_FMA7_RC18> reset CREF	**AB 77**	**CLOSED**
ASS-WAIT-FOR-UPDATE-CON UPDATE.CON(OK) \M-S AND O => ignore UPDATE.CON RSAP-ACTIVATE.req (ACCESS = All, Indication_Mode = unchanged) stop all timers, start T2	**AB 72**	**ABT-SAP-ACTIVATE[1)]**
ASS-WAIT-FOR-UPDATE-CON UPDATE.CON(LS/LR/IV) \M-S AND O => ignore UPDATE.CON RSAP-ACTIVATE.req (ACCESS = All, Indication_Mode = unchanged) stop all timers, start T2 LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LS/LR/IV)	**AB 78**	**ABT-SAP-ACTIVATE[1)]**

Connection Establishment at the Requester

Current State Event \Exit Condition => Action Taken	Transition	Next State
ASS-WAIT-FOR-UPDATE-CON T2 expired \M-S AND O => ignore UPDATE.CON RSAP-ACTIVATE.req (ACCESS = All, Indication_Mode = unchanged) stop all timers, start T2 LLI-Fault.ind <RC = LLI_FMA7_RC18)	**AB 79**	**ABT-SAP-ACTIVATE 1)**
ABT-WAIT-FOR-UPDATE-CON UPDATE.CON(OK) \M-S => ignore UPDATE.CON ABT.ind to LLI user <RC = ABT_RC2> send ABT_REQ_PDU <RC = ABT_RC2> UPDATE.req(low)) stop all timers, start T2	**AB 73**	**ABT-UPDATE 1)**
ABT-WAIT-FOR-UPDATE-CON UPDATE.CON(LR) \M-S => ignore UPDATE.CON ABT.ind to LLI user <RC = ABT_RC2> send ABT_REQ_PDU <RC = ABT_RC2> UPDATE.req(low)) stop all timers, start T2 LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LR)	**AB 80**	**ABT-UPDATE 1)**
ABT-WAIT-FOR-UPDATE-CON UPDATE.CON(LS/IV) \M-S AND D => ignore UPDATE.CON ABT.ind to LLI user <RC = ABT_RC2> reset CREF LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LS/IV)	**AB 87**	**CLOSED**
ABT-WAIT-FOR-UPDATE-CON UPDATE.CON(LS/IV) \M-S AND O => ignore UPDATE.CON ABT.ind to LLI user <RC = ABT_RC2> stop all timers, start T2 LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LS/IV) RSAP-ACTIVATE.req(Access = All, Indication_Mode unchanged)	**AB 88**	**ABT-SAP-ACTIVATE 1)**
ABT-WAIT-FOR-UPDATE-CON T2 expired \M-S => ABT.ind to LLI user <RC = ABT_RC2> send ABT_REQ_PDU <RC = ABT_RC2> UPDATE.req(low)) stop all timers, start T2 LLI-Fault.ind <RC = LLI_FMA7_RC18)	**AB 81**	**ABT-UPDATE 1)**

Connection Establishment at the Requester

Current State	**Transition**	**Next State**
Event \Exit Condition => Action Taken		
ABT-WAIT-FOR-UPDATE-CON	**AB 74**	**ABT-WAIT-FOR-UPDATE-CON**
Any FDL-Primitive except UPDATE.CON(OK/LS/LR/IV) \M-S => ignore FDL-Primitive		
ASS-WAIT-FOR-UPDATE-CON	**AB 75**	**ASS-WAIT-FOR-UPDATE-CON**
Any FDL-Primitive except UPDATE.CON(OK/LS/LR/IV) \M-S => ignore FDL-Primitive		

1) see state diagram for connection release

6.7.2.3 State Diagram for Connection Release

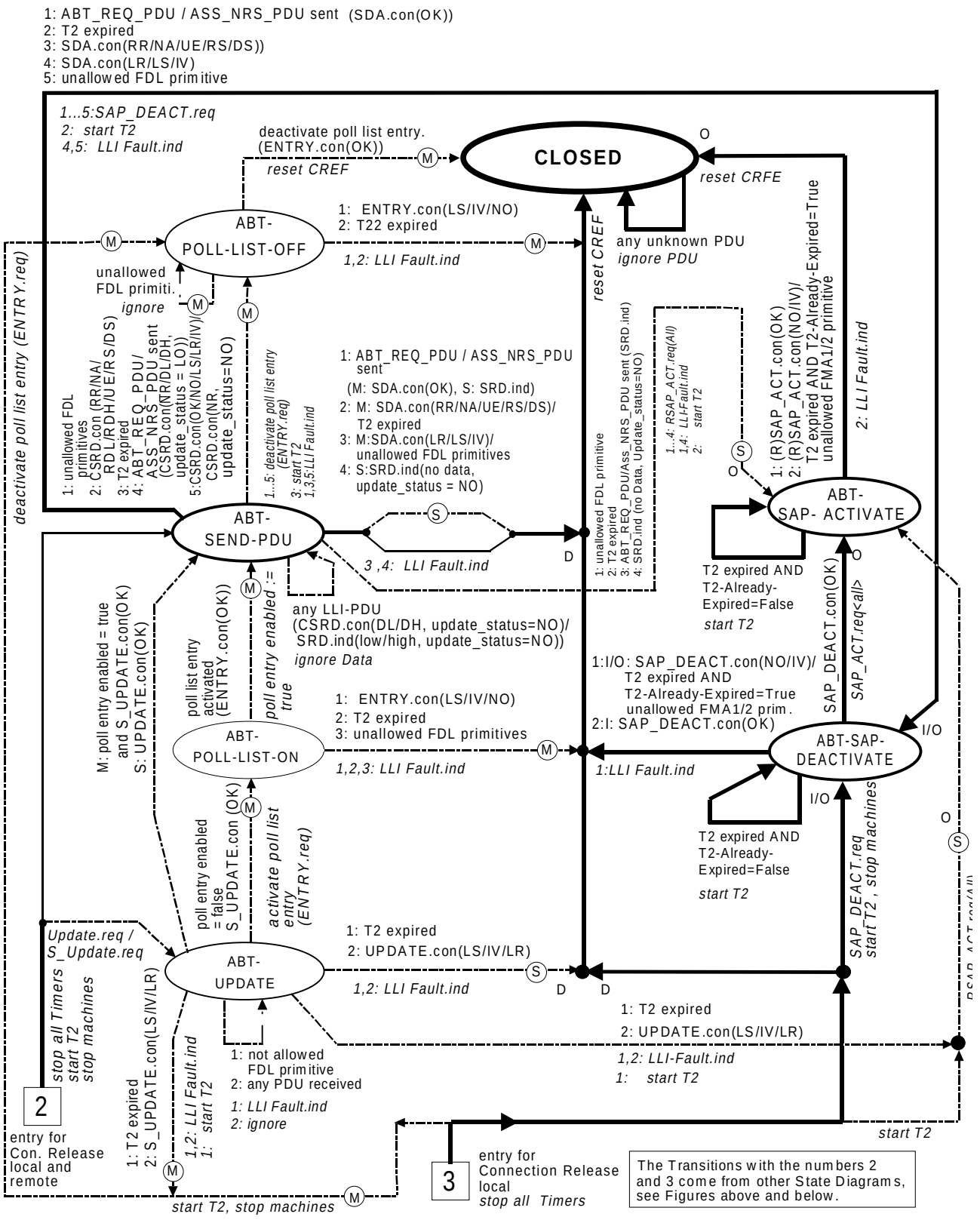


Figure 111. State Diagram for Connection Release

Description of State Transitions for Connection Release

Connection Release

Current State Event \Exit Condition => Action Taken	Transition	Next State
ABT-UPDATE unallowed FDL primitive \M-S => LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive>	**ABORT1**	**ABT-UPDATE**
ABT-UPDATE any LLI-PDU or unknown or faulty PDU received (SRD.ind(serv_class = low/high)/(C)SRD.con(DL/DH)) \M-S => ignore	**ABORT61**	**ABT-UPDATE**
ABT-UPDATE update buffer loaded (S_UPDATE.con(OK)) \M-S AND (poll entry enabled = false) AND Master => activate Poll List entry (ENTRY.req (remote address/remote LSAP/unlock))	**ABORT2**	**ABT-POLL-LIST-ON**
ABT-UPDATE update buffer loaded (UPDATE.con(OK)) \M-S AND Slave	**ABORT3**	**ABT-SEND-PDU**
ABT-UPDATE update buffer loaded (S_UPDATE.con (OK)) \M-S AND (poll entry enabled = true) AND Master	**ABORT4**	**ABT-SEND-PDU**
ABT-UPDATE T2 expired \M-S AND Slave AND D => LLI-Fault.ind <RC = LLI_FMA7_RC19> reset CREF	**ABORT5**	**CLOSED**
ABT-UPDATE error in loading the update buffer (UPDATE.con(LS/IV/LR)) \M-S AND Slave AND D => LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LS/IV/LR> reset CREF	**ABORT6**	**CLOSED**
ABT-UPDATE T2 expired \M-S AND Slave AND O => LLI-Fault.ind <RC = LLI_FMA7_RC19> start T2 RSAP_ACT.req(Access = All, Indication_mode = Unchanged)	**ABORT8**	**ABT-SAP-ACTIVATE**
ABT-UPDATE error in loading the update buffer (UPDATE.con(LS/IV/LR)) \M-S AND Slave AND O => LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LS/IV/LR> RSAP_ACT.req(Access = All, Indication_mode = Unchanged)	**ABORT9**	**ABT-SAP-ACTIVATE**

Connection Release

Current State Event \Exit Condition => Action Taken	Transition	Next State
ABT-UPDATE T2 expired \M-S AND Master => LLI-Fault.ind <RC = LLI_FMA7_RC19> start T2 deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock))	**ABORT11**	**ABT-POLL-LIST-OFF**
ABT-UPDATE error in loading the update buffer (S_UPDATE.con(LS/IV/LR)) \M-S AND Master => LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LS/IV/LR> deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock))	**ABORT12**	**ABT-POLL-LIST-OFF**
ABT-POLL-LIST-ON Poll List entry activated (ENTRY.con(OK)) \M-S AND Master => poll entry enabled := true	**ABORT14**	**ABT-SEND-PDU**
ABT-POLL-LIST-ON error in the activation of the Poll List entry (ENTRY.con(LS/IV/NO)) \M-S AND Master => LLI-Fault.ind <RC = LLI_FMA7_RC4, AD = LS/IV/NO> reset CREF	**ABORT15**	**CLOSED**
ABT-POLL-LIST-ON T2 expired \M-S AND Master => LLI-Fault.ind <RC = LLI_FMA7_RC19> reset CREF	**ABORT16**	**CLOSED**
ABT-POLL-LIST-ON unallowed FDL primitive \M-S AND Master => LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> reset CREF	**ABORT17**	**CLOSED**
ABT-SEND-PDU unallowed FDL primitive \M-S AND Master => deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive>	**ABORT18**	**ABT-POLL-LIST-OFF**
ABT-SEND-PDU T2 expired \M-S AND Master => deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) LLI-Fault.ind <RC = LLI_FMA7_RC19> start T2	**ABORT19**	**ABT-POLL-LIST-OFF**

Connection Release

Current State Event \Exit Condition => Action Taken	Transition	Next State
ABT-SEND-PDU ABT_REQ_PDU/ASS_NRS_PDU sent (SRD.ind(serv_class = low/high, update_status = LO)) \M-S AND Slave AND O => RSAP_ACT.req(Access = All, Indication_mode = Unchanged)	**ABORT20**	**ABT-SAP-ACTIVATE**
ABT-SEND-PDU ABT_REQ_PDU/ASS_NRS_PDU sent (SDA.con(OK)) \M-M AND (I OR O) => SAP_DEACT.req	**ABORT21**	**ABT-SAP-DEACTIVATE**
ABT-SEND-PDU T2 expired \M-S AND Slave AND O => RSAP_ACT.req(Access = All, Indication_mode = Unchanged) start T2	**ABORT22**	**ABT-SAP-ACTIVATE**
ABT-SEND-PDU T2 expired \M-M AND (I OR O) => SAP_DEACT.req start T2	**ABORT23**	**ABT-SAP-DEACTIVATE**
ABT-SEND-PDU error on sending the ABT_REQ_PDU (SDA.con(RR/NA/UE/RS/DS)) \M-M AND (I OR O) => SAP_DEACT.req	**ABORT24**	**ABT-SAP-DEACTIVATE**
ABT-SEND-PDU error on sending the ASS_NRS_PDU (SDA.con(RR/NA/UE/RS/DS)) \M-M AND O => SAP_DEACT.req	**ABORT59**	**ABT-SAP-DEACTIVATE**
ABT-SEND-PDU error on sending the ABT_REQ_PDU (SDA.con(LS/LR/IV)) \M-M AND (I OR O) => SAP_DEACT.req LLI-Fault.ind <RC = LLI_FMA7_RC12, AD = LS/LR/IV>	**ABORT25**	**ABT-SAP-DEACTIVATE**
ABT-SEND-PDU error on sending the ASS_NRS_PDU (SDA.con(LS/LR/IV)) \M-M AND O => SAP_DEACT.req LLI-Fault.ind <RC = LLI_FMA7_RC12, AD = LS/LR/IV>	**ABORT60**	**ABT-SAP-DEACTIVATE**
ABT-SEND-PDU ABT_REQ_PDU or ASS_NRS_PDU sent (SRD.ind(serv_class = low/high, update_status = LO)) \M-S AND Slave AND D => reset CREF	**ABORT26**	**CLOSED**

Connection Release

Current State Event \Exit Condition => Action Taken	Transition	Next State
ABT-SEND-PDU ABT_REQ_PDU or ASS_NRS_PDU sent (SDA.con(OK)) \M-M AND D => reset CREF	**ABORT27**	**CLOSED**
ABT-SEND-PDU error on sending the ABT_REQ_PDU or ASS_NRS_PDU (SDA.con(RR/NA/UE/RS/DS)) \M-M AND D => reset CREF	**ABORT28**	**CLOSED**
ABT-SEND-PDU T2 expired \M-S AND Slave AND D => reset CREF	**ABORT29**	**CLOSED**
ABT-SEND-PDU T2 expired \M-M AND D => reset CREF	**ABORT30**	**CLOSED**
ABT-SEND-PDU error on sending the ABT_REQ_PDU or ASS_NRS_PDU (SDA.con(LS/LR/IV)) \M-M AND D => LLI-Fault.ind <RC = LLI_FMA7_RC12, AD = LS/LR/IV> reset CREF	**ABORT31**	**CLOSED**
ABT-SEND-PDU unallowed FDL primitive \M-S AND Slave AND D => LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> reset CREF	**ABORT32**	**CLOSED**
ABT-SEND-PDU unallowed FDL primitive \M-M AND D => LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> reset CREF	**ABORT33**	**CLOSED**
ABT-SEND-PDU unallowed FDL primitive \M-S AND Slave AND O => LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> RSAP_ACT.req(Access = All, Indication_mode = Unchanged)	**ABORT56**	**ABT-SAP-ACTIVATE**
ABT-SEND-PDU unallowed FDL primitive \M-M AND (I or O) => LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive> SAP_DEACT.req	**ABORT57**	**ABT-SAP-DEACTIVATE**
ABT-POLL-LIST-OFF unallowed FDL primitive \M-S AND Master => LLI-Fault.ind <RC = LLI_FMA7_RC6, AD = code of the primitive>	**ABORT34**	**ABT-POLL-LIST-OFF**

Connection Release

Current State Event \Exit Condition => Action Taken	Transition	Next State
ABT-POLL-LIST-OFF any LLI-PDU or unknown or faulty PDU received \M-S AND Master => ignore	**ABORT62**	**ABT-POLL-LIST-OFF**
ABT-POLL-LIST-OFF Poll List entry deactivated (ENTRY.con(OK)) \M-S AND Master => poll entry enabled := false reset CREF	**ABORT35**	**CLOSED**
ABT-POLL-LIST-OFF error in the deactivation of the Poll List entry (ENTRY.con(LS/IV/NO)) \M-S AND Master => LLI-Fault.ind <RC = LLI_FMA7_RC5, AD = LS/IV/NO> reset CREF	**ABORT36**	**CLOSED**
ABT-POLL-LIST-OFF T2 expired \M-S AND Master => LLI-Fault.ind <RC = LLI_FMA7_RC19> reset CREF	**ABORT37**	**CLOSED**
ABT-SAP-DEACTIVATE SAP_DEACT.con(NO/IV) => LLI-Fault.ind <RC = LLI_FMA7_RC2, AD = error state> reset CREF	**ABORT38**	**CLOSED**
ABT-SAP-DEACTIVATE SAP_DEACT.con(OK) \M-M AND I => reset CREF	**ABORT39**	**CLOSED**
ABT-SAP-DEACTIVATE SAP_DEACT.con(OK) \M-M AND O => SAP_ACT.req(Access = All)	**ABORT41**	**ABT-SAP-ACTIVATE**
ABT-SAP-ACTIVATE (R)SAP_ACT.con(OK) \ => reset CREF	**ABORT43**	**CLOSED**
ABT-SAP-ACTIVATE (R)SAP_ACT.con(NO/IV) \ => LLI-Fault.ind <RC = LLI_FMA7_RC1, AD = error state> reset CREF	**ABORT44**	**CLOSED**

Connection Release

Current State Event \Exit Condition => Action Taken	Transition	Next State
CLOSED faulty or unknown PDU received \M-M OR M-S => ignore PDU	**ABORT45**	**CLOSED**
ABT-SAP-DEACTIVATE T2 expired \M-M AND (O OR I) AND T2-Already-Expired = TRUE => LLI-Fault.ind <RC = LLI_FMA7_RC19> reset CREF	**ABORT58**	**CLOSED**
ABT-SAP-DEACTIVATE T2 expired \M-M AND (O OR I) AND T2-Already-Expired = FALSE => start T2 T2-Already-Expired = TRUE	**ABORT63**	**ABT-SAP-DEACTIVATE**
ABT-SAP-ACTIVATE T2 expired \T2-Already-Expired = TRUE => LLI-Fault.ind <RC = LLI_FMA7_RC19> reset CREF	**ABORT47**	**CLOSED**
ABT-SAP-ACTIVATE T2 expired \T2-Already-Expired = FALSE => T2-Already-Expired = TRUE start T2	**ABORT65**	**ABT-SAP-ACTIVATE**
ABT-SEND-PDU error on sending the ABT_REQ_PDU (CSRD.con(L_status = RR/NA/RDL/RDH/UE/RS/DS)) OR ABT_REQ_PDU sent (CSRD.con(L_status = NR/DL/DH, update_status = LO)) OR ASS_NRS_PDU sent (CSRD.con(L_status = NR/DL/DH, update_status = LO)) \M-S AND Master => deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) ignore data if present	**ABORT48**	**ABT-POLL-LIST-OFF**
ABT-SEND-PDU error on sending the ABT_REQ_PDU (CSRD.con(L_status = OK/NO/LS/LR/IV)) \M-S AND Master => deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) LLI-Fault.ind <RC = LLI_FMA7_RC13, AD = OK/NO/LS/LR/IV>	**ABORT49**	**ABT-POLL-LIST-OFF**
ABT-SEND-PDU any LLI PDU or unknown or faulty PDU received (CSRD.con(L_status = DL/DH, update_status = NO)/ SRD.ind(serv_class = low/high, update_status = NO)) \M-S => ignore	**ABORT50**	**ABT-SEND-PDU**

Connection Release

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
ABT-SEND-PDU SRD.ind(no data, update_status = NO)) \M-S AND Slave AND D => LLI-Fault.ind <RC = LLI_FMA7_RC24> reset CREF	**ABORT51**	**CLOSED**
ABT-SEND-PDU SRD.ind(no data, update_status = NO)) \M-S AND Slave AND O => SAP_DEACT.req LLI-Fault.ind <RC = LLI_FMA7_RC24>	**ABORT52**	**ABT-SAP-ACTIVATE**
ABT-SEND-PDU CSRD.con(L_status = NR, update_status = NO)) \M-S AND Master => deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) LLI-Fault.ind <RC = LLI_FMA7_RC24>	**ABORT53**	**ABT-POLL-LIST-OFF**
ABT-SAP-DEACTIVATE unallowed FMA1/2 primitive received => LLI-Fault.ind <RC = LLI_FMA7_RC22> reset CREF	**ABORT54**	**CLOSED**
ABT-SAP-ACTIVATE unallowed FMA1/2 primitive received => LLI-Fault.ind <RC = LLI_FMA7_RC22> reset CREF	**ABORT55**	**CLOSED**

6.7.3 Data Transfer

6.7.3.1 State Diagram for Open at the Master

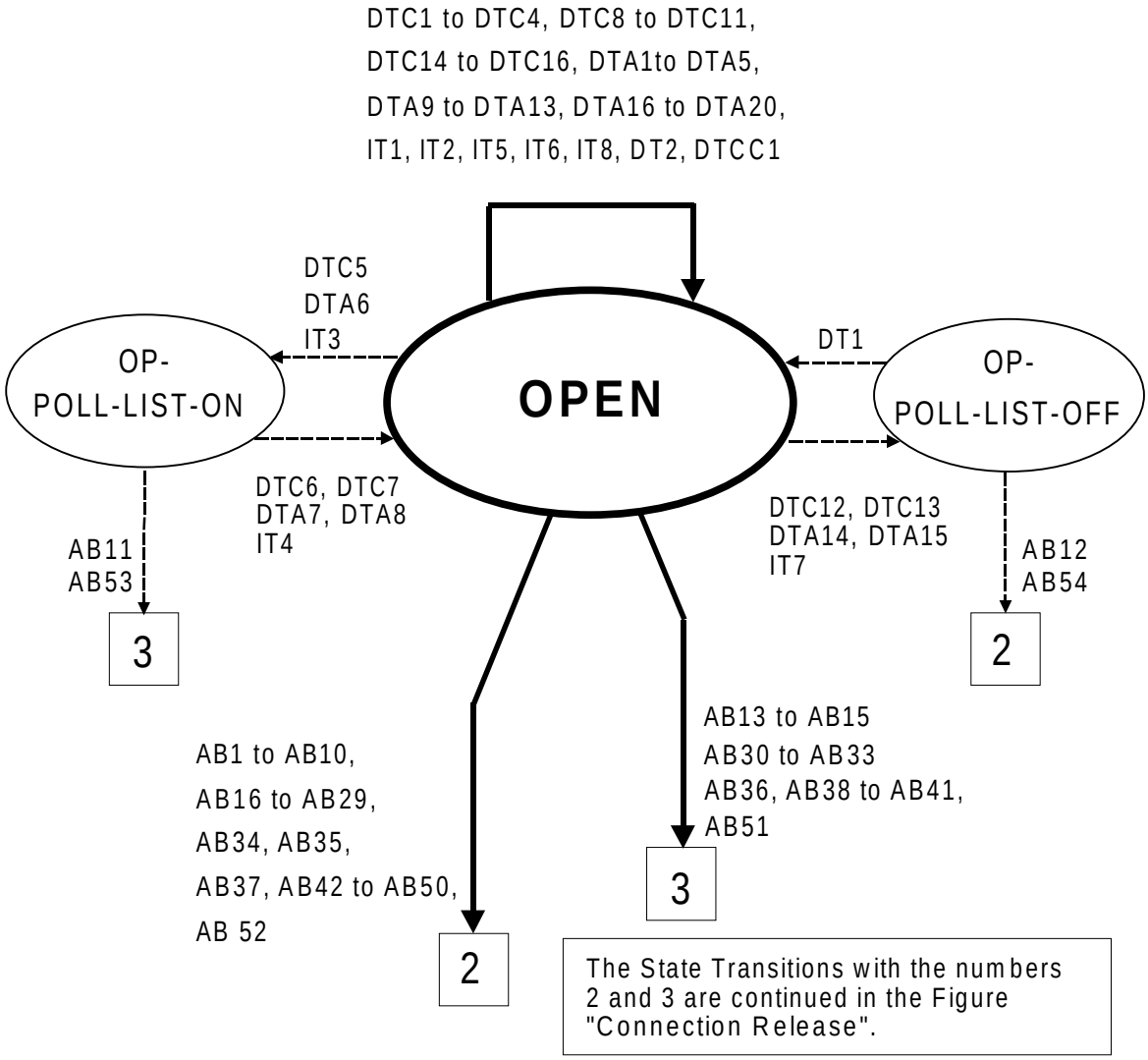


Figure 112. State Diagram for Open at the Master

Description of State Transitions for Open at the Master

Open at the Master

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
OPEN DTC.req from LLI user \(M-M OR acyc. M-S with SI) AND (SCC < max SCC) AND (ACI = 0) => start DTC Req.(invoke ID) SCC := SCC + 1	**DTC 1**	**OPEN**
OPEN DTC.req from LLI user \(M-M OR acyc. M-S with SI) AND (SCC < max SCC) AND (ACI > 0) => start DTC Req.(invoke ID) SCC := SCC + 1 start STimer	**DTC 2**	**OPEN**
OPEN DTC.req from LLI user \M-M AND (SCC ≥ max SCC) => ABT.ind to LLI user <RC = ABT_RC23> send ABT_REQ_PDU <RC = ABT_RC9> (SDA.req(low)) stop all timers, start T2, stop machines	**AB 1**	**ABT-SEND-PDU**[1)]
OPEN DTC.req from LLI user \acyc. M-S AND (SCC ≥ max SCC) => ABT.ind to LLI user <RC = ABT_RC23> send ABT_REQ_PDU <RC = ABT_RC9> (S_UPDATE.req) stop all timers, start T2, stop machines	**AB 2**	**ABT-UPDATE**[1)]
OPEN DTC.req from LLI user \acyc. M-S without SI AND (poll entry enabled = true) AND (SCC < max SCC) AND (ACI = 0) => start DTC Req.(invoke ID) SCC := SCC + 1	**DTC 3**	**OPEN**
OPEN DTC.req from LLI user \acyc. M-S without SI AND (poll entry enabled = true) AND (SCC < max SCC) AND (ACI > 0) => start DTC Req.(invoke ID) SCC := SCC + 1 start STimer	**DTC 4**	**OPEN**

1) see state diagram for connection release

Open at the Master

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN DTC.req from LLI user \acyc. M-S without SI AND (poll entry enabled = false) AND (SCC < max SCC) => activate Poll List entry (ENTRY.req(remote address/remote LSAP/unlock)) SCC := SCC + 1 Start T1	**DTC 5**	**OP-POLL-LIST-ON**
OP-POLL-LIST-ON Poll List entry activated (ENTRY.con(OK)) \acyc. M-S without SI AND (SCC = 1) AND (ACI = 0) => start DTC Req.(invoke ID) poll entry enabled := true Stop T1	**DTC 6**	**OPEN**
OP-POLL-LIST-ON Poll List entry activated (ENTRY.con(OK)) \acyc. M-S without SI AND (SCC = 1) AND (ACI > 0) => start DTC Req.(invoke ID) start STimer poll entry enabled := true Stop T1	**DTC 7**	**OPEN**
OPEN DTC Req. finished \(M-M OR acyc. M-S with SI) AND (ACI = 0) => SCC := SCC - 1	**DTC 8**	**OPEN**
OPEN DTC Req. finished \(M-M OR acyc. M-S with SI) AND (ACI > 0) => SCC := SCC - 1 start RTimer	**DTC 9**	**OPEN**
OPEN DTC Req. finished \acyc. M-S without SI AND ((SCC - 1) + SAC > 0) AND (ACI = 0) => SCC := SCC - 1	**DTC 10**	**OPEN**
OPEN DTC Req. finished \acyc. M-S without SI AND ((SCC - 1) + SAC > 0 OR IMA = true) AND (ACI > 0) => SCC := SCC - 1 start RTimer	**DTC 11**	**OPEN**
OPEN DTC Req. finished \acyc. M-S without SI AND ((SCC - 1) + SAC = 0) AND (ACI = 0) => SCC := SCC - 1 deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) Start T2	**DTC 12**	**OP-POLL-LIST-OFF**

Open at the Master

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN DTC Req. finished \acyc. M-S without SI AND ((SCC - 1) + SAC = 0 AND IMA = false) AND (ACI > 0) => SCC := SCC - 1 start RTimer deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) Start T2	**DTC 13**	**OP-POLL-LIST-OFF**
OPEN DTA.req from LLI user \(M-M OR acyc. M-S with SI) AND (SAC < max SAC) AND (ACI = 0) => start DTA Req. SAC := SAC + 1	**DTA 1**	**OPEN**
OPEN DTA.req from LLI user \(M-M OR acyc. M-S with SI) AND (SAC < max SAC) AND (ACI > 0) => start DTA Req. SAC := SAC + 1 start STimer	**DTA 2**	**OPEN**
OPEN DTA.req from LLI user \(M-M OR M-S) AND (SAC ≥ max SAC) => ignore DTA.req	**DTA 3**	**OPEN**
OPEN DTA.req from LLI user \acyc. M-S without SI AND (poll entry enabled = true) AND (SAC < max SAC) AND (ACI = 0) => start DTA Req. SAC := SAC + 1	**DTA 4**	**OPEN**
OPEN DTA.req from LLI user \acyc. M-S without SI AND (poll entry enabled = true) AND (SAC < max SAC) AND (ACI > 0) => start DTA Req. SAC := SAC + 1 start STimer	**DTA 5**	**OPEN**
OPEN DTA.req from LLI user \acyc. M-S without SI AND (poll entry enabled = false) AND (SAC < max SAC) => activate Poll List entry (ENTRY.req(remote address/remote LSAP/unlock)) SAC := SAC + 1 Start T1	**DTA 6**	**OP-POLL-LIST-ON**

Open at the Master

Current State Event \Exit Condition => Action Taken	Transition	Next State
OP-POLL-LIST-ON Poll List entry activated (ENTRY.con(OK)) \acyc. M-S without SI AND (SAC = 1) AND (ACI = 0) => start DTA Req. poll entry enabled := true Stop T1	**DTA 7**	**OPEN**
OP-POLL-LIST-ON Poll List entry activated (ENTRY.con(OK)) \acyc. M-S without SI AND (SAC = 1) AND (ACI > 0) => start DTA Req. start STimer poll entry enabled := true Stop T1	**DTA 8**	**OPEN**
OPEN DTA Req. finished \M-M OR M-S	**DTA 9**	**OPEN**
OPEN DTA_ACK_PDU received (SDA.ind(serv_class = low/high)/(C)SRD.con(L_status = DL/DH)/SRD.con(DL/DH)) \(M-M OR acyc. M-S with SI) AND (SAC > 0) AND (ACI = 0) => SAC := SAC - 1	**DTA 10**	**OPEN**
OPEN DTA_ACK_PDU received (SDA.ind(serv_class = low/high)/(C)SRD.con(L_status = DL/DH)/SRD.con(DL/DH)) \(M-M OR acyc. M-S with SI) AND (SAC > 0) AND (ACI > 0) => start RTimer SAC := SAC - 1	**DTA 11**	**OPEN**
OPEN DTA_ACK_PDU received ((C)SRD.con(L_status = DL/DH)/SRD.con(DL/DH)) \acyc. M-S without SI AND (SAC > 0) AND (SCC + (SAC - 1) > 0) AND (ACI = 0) => SAC = SAC - 1	**DTA 12**	**OPEN**
OPEN DTA_ACK_PDU received ((C)SRD.con(L_status = DL/DH)/SRD.con(DL/DH)) \acyc. M-S without SI AND (SAC > 0) AND (SCC + (SAC - 1) > 0 OR IMA = true) AND (ACI > 0) => start RTimer SAC = SAC - 1	**DTA 13**	**OPEN**

Open at the Master

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN DTA_ACK_PDU received ((C)SRD.con(L_status = DL/DH)/SRD.con(DL/DH)) \acyc. M-S without SI AND (SAC > 0) AND (SCC + (SAC - 1) = 0) AND (ACI = 0) => SAC := SAC - 1 deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) Start T2	**DTA 14**	**OP-POLL-LIST-OFF**
OPEN DTA_ACK_PDU received ((C)SRD.con(L_status = DL/DH)/SRD.con(DL/DH)) \acyc. M-S without SI AND (SAC > 0) AND (SCC + (SAC - 1) = 0 AND IMA = false) AND (ACI > 0) => start RTimer SAC := SAC - 1 deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) Start T2	**DTA 15**	**OP-POLL-LIST-OFF**
OPEN DTA_ACK_PDU received (SDA.ind(serv_class = low/high)) \M-M AND (SAC = 0) => ABT.ind to LLI user <RC = ABT_RC5> send ABT_REQ_PDU <RC = ABT_RC5> (SDA.req(low)) stop all timers, start T2, stop machines	**AB 3**	**ABT-SEND-PDU**[1)]
OPEN DTA_ACK_PDU received ((C)SRD.con(L_status = DL/DH)/SRD.con(DL/DH)) \M-S AND (SAC = 0) => ABT.ind to LLI user <RC = ABT_RC5> send ABT_REQ_PDU <RC = ABT_RC5> (S_UPDATE.req) stop all timers, start T2, stop machines	**AB 4**	**ABT-UPDATE**[1)]
OPEN DTC_REQ_PDU received (SDA.ind(serv_class = low)) \M-M AND (RCC < max RCC) AND (ACI = 0) => start DTC Res.(invoke ID) RCC := RCC + 1	**DTC 14**	**OPEN**
OPEN DTC_REQ_PDU received (SDA.ind(serv_class = low)) \M-M AND (RCC < max RCC) AND (ACI > 0) => start DTC Res.(invoke ID) RCC := RCC + 1 start RTimer	**DTC 15**	**OPEN**

1) see state diagram for connection release

Open at the Master

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN DTC_REQ_PDU received (SDA.ind(serv_class = low)) \M-M AND (RCC ≥ max RCC) => ABT.ind to LLI user <RC = ABT_RC6> send ABT_REQ_PDU <RC = ABT_RC6> (SDA.req(low)) stop all timers, start T2, stop machines	**AB 5**	**ABT-SEND-PDU**[1)]
OPEN DTC Res. finished \M-M => RCC := RCC - 1	**DTC 16**	**OPEN**
OPEN DTA_REQ_PDU received (SDA.ind(serv_class = low/high)/ (C)SRD.con(L_status = DL/DH)/SRD.con(DL/DH)) \(M-M OR M-S with SI) AND (RAC < max RAC) AND (ACI = 0) => start DTA Ack. RAC = RAC + 1	**DTA 16**	**OPEN**
OPEN DTA_REQ_PDU received (SDA.ind(serv_class = low/high)/(C)SRD.con(L_status = DL/DH)/SRD.con(DL/DH)) \(M-M OR M-S with SI) AND (RAC < max RAC) AND (ACI > 0) => start DTA Ack. RAC = RAC + 1 start RTimer	**DTA 17**	**OPEN**
OPEN DTA_REQ_PDU received (SDA.ind(serv_class = low/high)) \M-M AND (RAC ≥ max RAC) => ABT.ind to LLI user <RC = ABT_RC6> send ABT_REQ_PDU <RC = ABT_RC6> (SDA.req(low)) stop all timers, start T2, stop machines	**AB 6**	**ABT-SEND-PDU**[1)]
OPEN DTA_REQ_PDU received ((C)SRD.con(L_status = DL/DH)/SRD.con(DL/DH)) \(M-S with SI) AND (RAC ≥ max RAC) => ABT.ind to LLI user <RC = ABT_RC6> send ABT_REQ_PDU <RC = ABT_RC6> (S_UPDATE.req) stop all timers, start T2, stop machines	**AB 7**	**ABT-UPDATE**[1)]
OPEN DTA_REQ_PDU received ((C)SRD.con(L_status = DL/DH)/SRD.con(DL/DH)) \M-S without SI => ABT.ind to LLI user <RC = ABT_RC3> send ABT_REQ_PDU <RC = ABT_RC3> (S_UPDATE.req) stop all timers, start T2, stop machines	**AB 8**	**ABT-UPDATE**[1)]

1) see state diagram for connection release

Open at the Master

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN DTA Ack. finished \M-M OR M-S with SI => RAC := RAC - 1	**DTA 18**	**OPEN**
OPEN STimer expired \M-M OR acyc. M-S with SI => start STimer start Idle Req.	**IT 1**	**OPEN**
OPEN STimer expired \acyc. M-S without SI AND (poll entry enabled = true) => start STimer IMA := true start Idle Req.	**IT 2**	**OPEN**
OPEN STimer expired \acyc. M-S without SI AND (poll entry enabled = false) => start STimer IMA := true activate Poll List entry (ENTRY.req(remote address/remote LSAP/unlock)) Start T1	**IT 3**	**OP-POLL-LIST-ON**
OP-POLL-LIST-ON Poll List entry activated (ENTRY.con(OK)) \acyc. M-S without SI AND (IMA = true) => start Idle Req. start STimer poll entry enabled := true Stop T1	**IT 4**	**OPEN**
OPEN Idle Req. finished \ M-M OR acyc. M-S with SI	**IT 5**	**OPEN**
OPEN Idle Req. finished \acyc. M-S without SI AND (SCC + SAC > 0) => IMA := false	**IT 6**	**OPEN**
OPEN Idle Req. finished \acyc. M-S without SI AND (SCC + SAC = 0) => IMA := false deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) Start T2	**IT 7**	**OP-POLL-LIST-OFF**

1) see state diagram for connection release

Open at the Master

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN	**IT 8**	**OPEN**
IDLE_REQ_PDU received (SDA.ind(serv_class=low)/(C)SRD.con(L_status=DL)) \(M-M OR acyc. M-S) AND (ACI > 0) => start RTimer		
OPEN	**AB 9**	**ABT-SEND-PDU[1)]**
RTimer expired \M-M => ABT.ind to LLI user <RC = ABT_RC12> send ABT_REQ_PDU <RC = ABT_RC12> (SDA.req(low)) stop all timers, start T2, stop machines		
OPEN	**AB 10**	**ABT-UPDATE[1)]**
RTimer expired \acyc. M-S => ABT.ind to LLI user <RC = ABT_RC12> send ABT_REQ_PDU <RC = ABT_RC12> (S_UPDATE.req) stop all timers, start T2, stop machines		
OP-POLL-LIST-ON	**AB 11**	**CLOSED**
error in the activation of the Poll List entry (ENTRY.con(LS/NO/IV)) \acyc. M-S without SI => ABT.ind to LLI user <RC = LS/NO/IV, AD = ABT_AD2> LLI-Fault.ind <RC = LLI_FMA7_RC4, AD = LS/NO/IV> reset CREF		
OP-POLL-LIST-ON	**AB 53**	**CLOSED**
T1 expired \acyc. M-S without SI => LLI-Fault.ind <RC = LLI_FMA7_RC20> ABT.ind to LLI user <RC = ABT_RC9> reset		
CREFOP-POLL-LIST-OFF	**DT 1**	**OPEN**
Poll List entry deactivated (ENTRY.con(OK)) \acyc. M-S without SI => poll entry enabled := false Stop T2		
OP-POLL-LIST-OFF	**AB 12**	**ABT-UPDATE[1)]**
error in the deactivation of the Poll List entry (ENTRY.con(LS/NO/IV)) \acyc. M-S without SI => ABT.ind to LLI user <RC = LS/NO/IV, AD = ABT_AD3> LLI-Fault.ind <RC = LLI_FMA7_RC5, AD = LS/NO/IV> send ABT_REQ_PDU <RC = ABT_RC9> (S_UPDATE.req) stop all timers, start T2, stop machines		

1) see state diagram for connection release

Open at the Master

Current State Event \Exit Condition => Action Taken	Transition	Next State
OP-POLL-LIST-OFF T2 expired \acyc. M-S without SI => LLI-Fault.ind <RC = LLI_FMA7 RC20> send ABT_REQ_PDU <RC = ABT_RC9> (S_UPDATE.req) ABT.ind to LLI user <RC = ABT_RC9> stop all timers, start T2, stop machines	**AB 54**	**ABT-UPDATE**[1]
OPEN ABT_REQ_PDU received (SDA.ind(serv_class = low/high)) \M-M AND (I OR O) => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> stop all timers, start T2, stop machines SAP_DEACT.req	**AB 13**	**ABT-SAP-DEACTIVATE**[1]
OPEN ABT_REQ_PDU received (SDA.ind(serv_class = low/high)) \M-M AND D => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> reset CREF	**AB 14**	**CLOSED**
OPEN ABT_REQ_PDU received ((C)SRD.con(L_status = DL/DH)) \M-S => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> deactivate Poll List entry (ENTRY.req (remote address/remote LSAP/lock)) stop all timers, start T2, stop machines	**AB 15**	**ABT-POLL-LIST-OFF**[1]
OPEN ABT.req from LLI user \M-M => send ABT_REQ_PDU (SDA.req(low)) stop all timers, start T2, stop machines	**AB 16**	**ABT-SEND-PDU**[1]
OPEN ABT.req from LLI user \M-S => send ABT_REQ_PDU (S_UPDATE.req) stop all timers, start T2, stop machines	**AB 17**	**ABT-UPDATE**[1]
OPEN unknown or faulty LLI PDU received \M-M => ABT.ind to LLI user <RC = ABT_RC4> send ABT_REQ_PDU <RC = ABT_RC4> (SDA.req(low)) stop all timers, start T2, stop machines	**AB 18**	**ABT-SEND-PDU**[1]

1) see state diagram for connection release

Open at the Master

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN ASS_REQ_PDU received (SDA.ind(serv_class = low/high)) OR ASS_NRS_PDU received (SDA.ind(serv_class = low/high)) OR ASS_RES_PDU received (SDA.ind(serv_class = low/high)) OR DTU_REQ_PDU received (SDA.ind(serv_class = low/high)) \M-M => ABT.ind to LLI user <RC = ABT_RC3> send ABT_REQ_PDU <RC = ABT_RC3> (SDA.req(low)) stop all timers, start T2, stop machines	**AB 19**	**ABT-SEND-PDU**[1)]
OPEN IDLE_REQ_PDU received (SDA.ind(serv_class = high)) OR DTC_REQ_PDU received (SDA.ind(serv_class = high)) OR DTC_RES_PDU received (SDA.ind(serv_class = high)) \M-M => ABT.ind to LLI user <RC = ABT_RC8> send ABT_REQ_PDU <RC = ABT_RC8> (SDA.req(low)) stop all timers, start T2, stop machines	**AB 20**	**ABT-SEND-PDU**[1)]
OPEN ASS.req OR ASS.res OR DTU.req from LLI user \M-M => ABT.ind to LLI user <RC = ABT_RC19, AD = code of the primitive> send ABT_REQ_PDU <RC = ABT_RC9> (SDA.req(low)) LLI-Fault.ind <RC = LLI_FMA7_RC11> stop all timers, start T2, stop machines	**AB 21**	**ABT-SEND-PDU**[1)]
OPEN unknown LLI primitive \M-M => ABT.ind to LLI user <RC = ABT_RC17> send ABT_REQ_PDU <RC = ABT_RC9> (SDA.req(low)) LLI-Fault.ind <RC = LLI_FMA7_RC9> stop all timers, start T2, stop machines	**AB 22**	**ABT-SEND-PDU**[1)]
OPEN Layer 2 primitive AND NOT(SDA.ind OR SDA.con) \M-M => ABT.ind to LLI user <RC = ABT_RC15, AD = code of the primitive> send ABT_REQ_PDU <RC = ABT_RC9> (SDA.req(low)) LLI-Fault.ind <RC = LLI_FMA7_RC7, AD = code of the primitive> stop all timers, start T2, stop machines	**AB 23**	**ABT-SEND-PDU**[1)]
OPEN unknown or faulty LLI PDU received \M-S => ABT.ind to LLI user <RC = ABT_RC4> send ABT_REQ_PDU <RC = ABT_RC4> (S_UPDATE.req) stop all timers, start T2, stop machines	**AB 24**	**ABT-UPDATE**[1)]

1) see state diagram for connection release

Open at the Master

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN ASS_REQ_PDU OR ASS_NRS_PDU OR ASS_RES_PDU OR DTC_REQ_PDU received OR DTU_REQ_PDU OR DTC_REQ_PDU received ((C)SRD.con(L_status = DL/DH)/SRD.con(DL/DH)) \M-S => ABT.ind to LLI user <RC = ABT_RC3> send ABT_REQ_PDU <RC = ABT_RC3> (S_UPDATE.req) stop all timers, start T2, stop machines	**AB 25**	**ABT-UPDATE**[1)]
OPEN IDLE_REQ_PDU OR DTC_RES_PDU received ((C)SRD.con(L_status = DH)/SRD.con(DH)) \M-S => ABT.ind to LLI user <RC = ABT_RC8> send ABT_REQ_PDU <RC = ABT_RC8> (S_UPDATE.req) stop all timers, start T2, stop machines	**AB 26**	**ABT-UPDATE**[1)]
OPEN ASS.req OR ASS.res OR DTC.res OR DTU.req from LLI user \M-S => ABT.ind to LLI user <RC = ABT_RC19, AD = code of the primitive> send ABT_REQ_PDU <RC = ABT_RC9> (S_UPDATE.req) LLI-Fault.ind <RC = LLI_FMA7_RC11> stop all timers, start T2, stop machines	**AB 27**	**ABT-UPDATE**[1)]
OPEN unknown LLI primitive \M-S => ABT.ind to LLI user <RC = ABT_RC17> send ABT_REQ_PDU <RC = ABT_RC9> (S_UPDATE.req) LLI-Fault.ind <RC = LLI_FMA7_RC9> stop all timers, start T2, stop machines	**AB 28**	**ABT-UPDATE**[1)]
OPEN (Layer 2 primitive AND NOT ((C)SRD.con OR S_UPDATE.con)) \M-S => ABT.ind to LLI user <RC = ABT_RC15, AD = code of the primitive> send ABT_REQ_PDU <RC = ABT_RC9> (S_UPDATE.req) LLI-Fault.ind <RC = LLI_FMA7_RC7, AD = code of the primitive> stop all timers, start T2, stop machines	**AB 29**	**ABT-UPDATE**[1)]
OPEN receipt without data ((C)SRD.con(L_status = NR, update_status = LO)) \M-S	**DT 2**	**OPEN**
OPEN error on sending (SDA.con(UE/RS/DS)) \M-M AND D => ABT.ind to LLI user <RC = UE/RS/DS, AD = ABT_AD4> reset CREF	**AB 30**	**CLOSED**

1) see state diagram for connection release

Open at the Master

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN error on sending (SDA.con(UE/RS/DS)) \M-M AND (I OR O) => ABT.ind to LLI user <RC = UE/RS/DS, AD = ABT_AD4> stop all timers, start T2, stop machines SAP_DEACT.req	**AB 31**	**ABT-SAP-DEACTIVATE**[1)]
OPEN error on sending (SDA.con(LS/IV)) \M-M AND D => ABT.ind to LLI user <RC = LS/IV, AD = ABT_AD4> LLI-Fault.ind <RC = LLI_FMA7_RC12, AD = LS/IV> reset CREF	**AB 32**	**CLOSED**
OPEN error on sending (SDA.con(LS/IV)) \M-M AND (I OR O) => ABT.ind to LLI user <RC = LS/IV, AD = ABT_AD4> LLI-Fault.ind <RC = LLI_FMA7_RC12, AD = LS/IV> stop all timers, start T2, stop machines SAP_DEACT.req	**AB 33**	**ABT-SAP-DEACTIVATE**[1)]
OPEN error on sending (SDA.con(RR/NA)) \M-M => ABT.ind to LLI user <RC = RR/NA, AD = ABT_AD4> send ABT_REQ_PDU <RC = RR/NA> (SDA.req(low)) stop all timers, start T2, stop machines	**AB 34**	**ABT-SEND-PDU**[1)]
OPEN error on sending (SDA.con(LR)) \M-M => ABT.ind to LLI user <RC = LR, AD = ABT_AD4> send ABT_REQ_PDU <RC = LR> (SDA.req(low)) LLI-Fault.ind <RC = LLI_FMA7_RC12, AD = LR> stop all timers, start T2, stop machines	**AB 35**	**ABT-SEND-PDU**[1)]
OPEN error in loading the update buffer (S_UPDATE.con(LS/IV)) \M-S => ABT.ind to LLI user <RC = LS/IV, AD = ABT_AD1> LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LS/IV> deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) stop all timers, start T2, stop machines	**AB 36**	**ABT-POLL-LIST-OFF**[1)]

1) see state diagram for connection release

Open at the Master

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN	**AB 37**	**ABT-UPDATE**[1)]
error in loading the update buffer (S_UPDATE.con(LR)) \M-S => ABT.ind to LLI user <RC = LR, AD = ABT_AD1> send ABT_REQ_PDU <RC = LR> (S_UPDATE.req(low)) LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LR> stop all timers, start T2, stop machines		
OPEN	**AB 38**	**ABT-POLL-LIST-OFF**[1)]
error on sending (CSRD.con(L_status = UE/RS/DS)) \M-S => ABT.ind to LLI user <RC = UE/RS/DS, AD = ABT_AD5> deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) stop all timers, start T2, stop machines		
OPEN	**AB 39**	**ABT-POLL-LIST-OFF**[1)]
error on sending (SRD.con(UE/RS/DS)) \M-S => ABT.ind to LLI user <RC = UE/RS/DS, AD = ABT_AD6> deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) stop all timers, start T2, stop machines		
OPEN	**AB 40**	**ABT-POLL-LIST-OFF**[1)]
error on sending (CSRD.con(L_status = LS/IV/OK/NO)) \M-S => ABT.ind to LLI user <RC = LS/IV/OK/NO, AD = ABT_AD5> LLI-Fault.ind <RC = LLI_FMA7_RC13, AD = LS/IV/OK/NO> deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) stop all timers, start T2, stop machines		
OPEN	**AB 41**	**ABT-POLL-LIST-OFF**[1)]
error on sending (SRD.con(LS/IV)) \M-S => ABT.ind to LLI user <RC = LS/IV, AD = ABT_AD6> LLI-Fault.ind <RC = LLI_FMA7_RC14, AD = LS/IV> deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) stop all timers, start T2, stop machines		
OPEN	**AB 42**	**ABT-UPDATE**[1)]
error on sending (CSRD.con(L_status = RR/NA/RDL/RDH)) \M-S => ABT.ind to LLI user <RC = RR/NA/RDL/RDH, AD = ABT_AD5> send ABT_REQ_PDU <RC = RR/NA/RDL/RDH> (S_UPDATE.req(low)) stop all timers, start T2, stop machines ignore data if present		

1) see state diagram for connection release

Open at the Master

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN	**AB 43**	**ABT-UPDATE**[1)]
error on sending (SRD.con(RR/NA/RDL/RDH)) \M-S => ABT.ind to LLI user <RC = RR/NA/RDL/RDH, AD = ABT_AD6> send ABT_REQ_PDU <RC = RR/NA/RDL/RDH> (S_UPDATE.req(low)) stop all timers, start T2, stop machines ignore data if present		
OPEN	**AB 44**	**ABT-UPDATE**[1)]
error on sending (CSRD.con(L_status = LR)) \M-S => ABT.ind to LLI user <RC = LR, AD = ABT_AD5> send ABT_REQ_PDU <RC = LR> (S_UPDATE.req(low)) LLI-Fault.ind <RC = LLI_FMA7_RC13, AD = LR> stop all timers, start T2, stop machines		
OPEN	**AB 45**	**ABT-UPDATE**[1)]
error on sending (SRD.con(LR)) \M-S => ABT.ind to LLI user <RC = LR, AD = ABT_AD6> send ABT_REQ_PDU <RC = LR> (S_UPDATE.req(low)) LLI-Fault.ind <RC = LLI_FMA7_RC14, AD = LR> stop all timers, start T2, stop machines		
OPEN	**AB 46**	**ABT-SEND-PDU**[1)]
DTC_RES_PDU received (SDA.ind(serv_class = low/high)) OR DTC.res from LLI user \M-M AND (invoke ID = unknown) AND (LLI-SAP=0) => ABT.ind to LLI user <RC = ABT_RC7> send ABT_REQ_PDU <RC = ABT_RC7> (SDA.req(low)) stop all timers, start T2, stop machines		
OPEN	**AB 47**	**ABT-UPDATE**[1)]
DTC_RES_PDU received ((C)SRD.con(L_status = DL/DH)) OR DTC.res from LLI user \M-S AND (invoke ID = unknown) AND (LLI-SAP=0) => ABT.ind to LLI user <RC = ABT_RC7> send ABT_REQ_PDU <RC = ABT_RC7> (S_UPDATE.req) stop all timers, start T2, stop machines		
OPEN	**DTCC 1**	**OPEN**
DTC.req from LLI user \cyc. M-S AND (status DTCC = DTCC-WAIT-FOR-REQ) AND (Read OR Write) => start DTCC Req.		

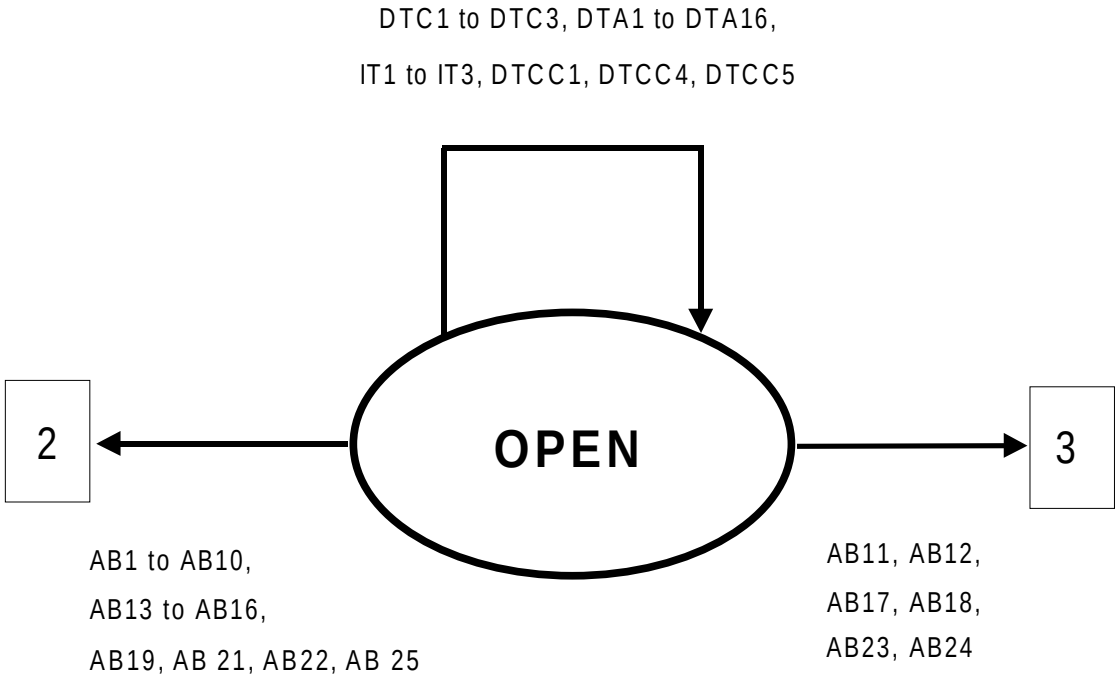
1) see state diagram for connection release

Open at the Master

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN T3 expired \cyc. M-S => send ABT_REQ_PDU <RC = ABT_RC11> (S_UPDATE.req) ABT.ind to LLI user <RC = ABT_RC11> stop all timers, start T2, stop machines	**AB 48**	**ABT-UPDATE[1)]**
OPEN IDLE_REQ_PDU received (SDA.ind(serv_class = low)) \M-M AND (ACI = 0) => send ABT_REQ_PDU <RC = ABT_RC3> (SDA.req(low)) ABT.ind to LLI user <RC = ABT_RC3> stop all timers, start T2, stop machines	**AB 49**	**ABT-SEND-PDU[1)]**
OPEN IDLE_REQ_PDU received ((C)SRD.con(L_status = DL)) \(acyc. M-S AND (ACI = 0)) OR (cyc. M-S) => send ABT_REQ_PDU <RC = ABT_RC3> (S_UPDATE.req) ABT.ind to LLI user <RC = ABT_RC3> stop all timers, start T2, stop machines	**AB 50**	**ABT-UPDATE[1)]**
OPEN DTA.req from LLI user \cyc. M-S AND (SAC < max SAC) => start DTA Req. SAC := SAC + 1	**DTA 19**	**OPEN**
OPEN DTA_ACK_PDU received ((C)SRD.con(L_status = DL/DH)/SRD.con(DL/DH)) \cyc. M-S AND (SAC > 0) => SAC := SAC - 1	**DTA 20**	**OPEN**
OPEN receipt without data ((C)SRD.con(L_status = NR, update_status = NO)) \M-S => ABT.ind to LLI user <RC = ABT_RC25> LLI-Fault.ind <RC = LLI_FMA7_RC24> deactivate Poll List entry (ENTRY.req(remote address/remote LSAP/lock)) stop all timers, start T2, stop machines	**AB 51**	**ABT-POLL-LIST-OFF[1)]**
OPEN DTC.req from LLI user \cyc. M-S AND (status DTCC = DTCC-WAIT-FOR-REQ) AND NOT (Read OR Write) => send ABT_REQ_PDU <RC = ABT_RC9> (S_UPDATE.req) ABT.ind to LLI user <RC = ABT_RC27> stop all timers, start T2, stop machines	**AB 52**	**ABT-UPDATE[1)]**

1) see state diagram for connection release

6.7.3.2 State Diagram for Open at the Slave



The Transitions with the numbers 2 and 3 are continued in theFigure "Connection Release".

Figure 113. State Diagram for Open at the Slave

Description of State Transitions for Open at the Slave

Open at the Slave

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN DTA.req from LLI user \acyc. M-S with SI AND (SAC < max SAC) AND (ACI = 0) => start DTA Req. SAC := SAC + 1	**DTA 1**	**OPEN**
OPEN DTA.req from LLI user \acyc. M-S with SI AND (SAC < max SAC) AND (ACI > 0) => start DTA Req. SAC := SAC + 1 start STimer	**DTA 2**	**OPEN**
OPEN DTA.req from LLI user \M-S with SI AND (SAC < max SAC) => ignore DTA.req	**DTA 3**	**OPEN**
OPEN DTA.req from LLI user \M-S without SI => send ABT_REQ_PDU <RC = ABT_RC9> (UPDATE.req(low)) ABT. ind to LLI user <RC = ABT_RC19,AD = code of the primitive> LLI-Fault.ind <RC = LLI_FMA7_RC11, AD = code of the primitive> stop all timers, start T2, stop machines	**AB 1**	**ABT-UPDATE**[1)]
OPEN DTA_ACK_PDU received (SRD.ind(serv_class = low/high)) \acyc. M-S with SI AND (SAC > 0) AND (ACI = 0) => SAC := SAC - 1	**DTA 4**	**OPEN**
OPEN DTA_ACK_PDU received (SRD.ind(serv_class = low/high)) \acyc. M-S with SI AND (SAC > 0) AND (ACI > 0) => start RTimer SAC := SAC - 1	**DTA 5**	**OPEN**
OPEN DTA_ACK_PDU received (SRD.ind(serv_class = low/high)) \M-S with SI AND (SAC = 0) => send ABT_REQ_PDU <RC = ABT_RC5> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC5> stop all timers, start T2, stop machines	**AB 2**	**ABT-UPDATE**[1)]

1) see state diagram for connection release

Open at the Slave

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN DTA_ACK_PDU received (SRD.ind(serv_class = low/high)) \M-S without SI => send ABT_REQ_PDU <RC = ABT_RC3> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC3> stop all timers, start T2, stop machines	**AB 3**	**ABT-UPDATE**[1)]
OPEN DTA Req. finished \M-S with SI	**DTA 6**	**OPEN**
OPEN DTC_REQ_PDU received (SRD.ind(serv_class = low)) \acyc. M-S AND (RCC < max RCC) AND (ACI = 0) => start DTC Res. (invoke ID) RCC := RCC + 1	**DTC 1**	**OPEN**
OPEN DTC_REQ_PDU received (SRD.ind(serv_class = low)) \acyc. M-S AND (RCC < max RCC) AND (ACI > 0) => start DTC Res. (invoke ID) RCC := RCC + 1 start RTimer	**DTC 2**	**OPEN**
OPEN DTC_REQ_PDU received (SRD.ind(serv_class = low)) \acyc. M-S AND (RCC < max RCC) => send ABT_REQ_PDU <RC = ABT_RC6> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC6> stop all timers, start T2, stop machines	**AB 4**	**ABT-UPDATE**[1)]
OPEN DTC Res. finished \acyc. M-S => RCC := RCC - 1	**DTC 3**	**OPEN**
OPEN DTA_REQ_PDU received (SRD.ind(serv_class = low/high)) \acyc. M-S AND (RAC < max RAC) AND (ACI = 0) => start DTA Ack. RAC := RAC + 1	**DTA 7**	**OPEN**
OPEN DTA_REQ_PDU received (SRD.ind(serv_class = low/high)) \acyc. M-S AND (RAC < max RAC) AND (ACI > 0) => start DTA Ack. RAC := RAC + 1 start RTimer	**DTA 8**	**OPEN**

1) see state diagram for connection release

Open at the Slave

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN DTA_REQ_PDU received (SRD.ind(serv_class = low/high)) \M-S AND (RAC < max RAC) => send ABT_REQ_PDU <RC = ABT_RC6> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC6> stop all timers, start T2, stop machines	**AB 5**	**ABT-UPDATE**[1]
OPEN DTA Ack. finished \M-S => RAC := RAC - 1	**DTA 9**	**OPEN**
OPEN STimer expired \acyc. M-S => start STimer start Idle Req.	**IT 1**	**OPEN**
OPEN Idle Req. finished \acyc. M-S	**IT 2**	**OPEN**
OPEN IDLE_REQ_PDU received (SRD.ind(serv_class = low)) \acyc. M-S AND (ACI > 0) => start RTimer	**IT 3**	**OPEN**
OPEN RTimer expired \acyc. M-S => send ABT_REQ_PDU <RC = ABT_RC12> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC12> stop all timers, start T2, stop machines	**AB 6**	**ABT-UPDATE**[1]
OPEN DTC_RES_PDU received (SRD.ind(serv_class = low/high)) OR ASS_REQ_PDU received (SRD.ind(serv_class = low/high)) OR ASS_RES_PDU received (SRD.ind(serv_class = low/high)) OR ASS_NRS_PDU received (SRD.ind(serv_class = low/high)) OR DTU_REQ_PDU received (SRD.ind(serv_class = low/high)) \M-S => send ABT_REQ_PDU <RC = ABT_RC3> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC3> stop all timers, start T2, stop machines	**AB 7**	**ABT-UPDATE**[1]
OPEN IDLE_REQ_PDU received (SRD.ind(serv_class = high)) OR DTC_REQ_PDU received (SRD.ind(serv_class = high)) \M-S => send ABT_REQ_PDU <RC = ABT_RC8> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC8> stop all timers, start T2, stop machines	**AB 8**	**ABT-UPDATE**[1]

1) see state diagram for connection release

Open at the Slave

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN unknown or faulty LLI PDU received \M-S => send ABT_REQ_PDU <RC = ABT_RC4> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC4> stop all timers, start T2, stop machines	**AB 9**	**ABT-UPDATE**[1)]
OPEN DTC.res from LLI user \M-S AND (invoke ID = unknown) AND (LLI-SAP=0) => send ABT_REQ_PDU <RC = ABT_RC7> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC7> stop all timers, start T2, stop machines	**AB 10**	**ABT-UPDATE**[1)]
OPEN ABT_REQ_PDU received (SRD.ind(serv_class = low/high)) \M-S AND O => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> stop all timers, start T2, stop machines RSAP_ACT.req(Access = All, Indication_mode = Unchanged)	**AB 11**	**ABT-SAP-ACTIVATE**[1)]
OPEN ABT_REQ_PDU received (SRD.ind(serv_class = low/high)) \M-S AND D => ABT.ind to LLI user <RC = RC out of ABT_REQ_PDU> reset CREF	**AB 12**	**CLOSED**
OPEN ABT.req from LLI user \M-S => send ABT_REQ_PDU (UPDATE.req(low)) stop all timers, start T2, stop machines	**AB 13**	**ABT-UPDATE**[1)]
OPEN Layer 2 primitive AND NOT(UPDATE.con OR SRD.ind) \M-S => send ABT_REQ_PDU <RC = ABT_RC9> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC15, AD = code of the primitive> LLI-Fault.ind <RC = LLI_FMA7_RC7, AD = code of the primitive> stop all timers, start T2, stop machines	**AB 14**	**ABT-UPDATE**[1)]
OPEN DTC.req OR ASS.req OR ASS.res from LLI user \M-S => send ABT_REQ_PDU <RC = ABT_RC9> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC19, AD = code of the primitive> LLI-Fault.ind <RC = LLI_FMA7_RC11, AD = code of the primitive> stop all timers, start T2, stop machines	**AB 15**	**ABT-UPDATE**[1)]

1) see state diagram for connection release

Open at the Slave

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN unknown LLI primitive \M-S => send ABT_REQ_PDU <RC = ABT_RC9>, (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC17> LLI-Fault.ind <RC = LLI_FMA7_RC9> stop all timers, start T2, stop machines	**AB 16**	**ABT-UPDATE**[1)]
OPEN error in loading the update buffer (UPDATE.con(LS/IV)) \M-S AND D => ABT.ind to LLI user <RC = LS/IV, AD = ABT_AD1> LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LS/IV> reset CREF	**AB 17**	**CLOSED**
OPEN error in loading the update buffer (UPDATE.con(LS/IV)) \M-S AND O => ABT.ind to LLI user <RC = LS/IV, AD = ABT_AD1> LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LS/IV> stop all timers, start T2, stop machines RSAP_ACT.req(Access = All, Indication_mode = Unchanged)	**AB 18**	**ABT-SAP-ACTIVATE**[1)]
OPEN error in loading the update buffer (UPDATE.con(LR)) \M-S => ABT.ind to LLI user <RC = LR, AD = ABT_AD1> send ABT_REQ_PDU <RC = LR> (UPDATE.req(low)) LLI-Fault.ind <RC = LLI_FMA7_RC3, AD = LR> stop all timers, start T2, stop machines	**AB 19**	**ABT-UPDATE**[1)]
OPEN SRD.ind(no data, update_status = LO/HI) \cyc. M-S AND (status DTCC = DTCC-WAIT-FOR-REQ-PDU) => ignore	**DTCC 1**	**OPEN**
OPEN DTC_REQ_PDU received (SRD.ind(serv_class = low)) \cyc. M-S AND (status DTCC = DTCC-WAIT-FOR-REQ-PDU) AND (CCI = 0) AND (Read OR Write) => start DTCC Res.	**DTCC 4**	**OPEN**
OPEN DTC_REQ_PDU received (SRD.ind(serv_class = low)) \cyc. M-S AND (status DTCC = DTCC-WAIT-FOR-REQ-PDU) AND (CCI > 0) AND (Read OR Write) => start DTCC Res., start T3	**DTCC 5**	**OPEN**

1) see state diagram for connection release

Open at the Slave

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN DTA_REQ_PDU received (SRD.ind(serv_class = low/high)) \cyc. M-S AND (RAC < max RAC) AND (CCI = 0) => start DTA Ack. RAC := RAC + 1	**DTA 10**	**OPEN**
OPEN DTA_REQ_PDU received (SRD.ind(serv_class = low/high)) \cyc. M-S AND (RAC < max RAC) AND (CCI > 0) AND (Status DTCC # DTCC-WAIT-FOR-REQ-PDU) => start DTA Ack. RAC := RAC + 1 start T3	**DTA 11**	**OPEN**
OPEN DTA_REQ_PDU received (SRD.ind(serv_class = low/high)) \cyc. M-S AND (RAC < max RAC) AND (CCI > 0) AND (Status DTCC = DTCC-WAIT-FOR-REQ-PDU) => start DTA Ack., RAC := RAC + 1	**DTA 15**	**OPEN**
OPEN DTA.req from LLI user \cyc. M-S with SI AND (SAC < max SAC) => start DTA Req., SAC := SAC + 1	**DTA 12**	**OPEN**
OPEN DTA_ACK_PDU received (SRD.ind(serv_class = low/high)) \cyc. M-S with SI AND (SAC > 0) AND (CCI = 0) => SAC := SAC - 1	**DTA 13**	**OPEN**
OPEN DTA_ACK_PDU received (SRD.ind(serv_class = low/high)) \cyc. M-S with SI AND (SAC > 0) AND (CCI > 0) AND (Status DTCC # DTCC-WAIT-FOR-REQ-PDU) => start T3 SAC := SAC - 1	**DTA 14**	**OPEN**
OPEN DTA_ACK_PDU received (SRD.ind(serv_class = low/high)) \cyc. M-S with SI AND (SAC > 0) AND (CCI > 0) AND (Status DTCC = DTCC-WAIT-FOR-REQ-PDU) =>SAC := SAC - 1	**DTA 16**	**OPEN**
OPEN T3 expired \cyc. M-S => send ABT_REQ_PDU <RC = ABT_RC11> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC11> stop all timers, start T2, stop machines	**AB 21**	**ABT-UPDATE**[1)]

1) see state diagram for connection release

Open at the Slave

Current State Event \Exit Condition => Action Taken	Transition	Next State
OPEN IDLE_REQ_PDU received (SRD.ind(serv_class = low)) \(acyc. M-S AND (ACI = 0)) OR (cyc. M-S) => send ABT_REQ_PDU <RC = ABT_RC3> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC3> stop all timers, start T2, stop machines	**AB 22**	**ABT-UPDATE**[1]
OPEN SRD.ind(no data, update_status = NO) \M-S AND D => ABT.ind to LLI user <RC = ABT_RC25> LLI-Fault.ind <RC = LLI_FMA7_RC24> reset CREF	**AB 23**	**CLOSED**
OPEN SRD.ind(no data, update_status = NO) \M-S AND O => ABT.ind to LLI user <RC = ABT_RC25> LLI-Fault.ind <RC = LLI_FMA7_RC24> stop all timers, start T2, stop machines RSAP_ACT.req(Access = All, Indication_mode = Unchanged)	**AB 24**	**ABT-SAP-ACTIVATE**[1]
OPEN DTC_REQ_PDU received (SRD.ind (serv_class = low)) \cyc. M-S AND NOT (Read or Write) AND status DTCC = DTCC-WAIT-FOR-REQ-PDU => ABT.ind to LLI user <RC = ABT_RC27> send ABT_REQ_PDU <RC = ABT_RC27> (update.req(low)) stop all timers, start T2, stop machines	**AB 25**	**ABT-UPDATE**[1]

1) see state diagram for connection release

6.7.3.3 State Diagram for DTC Requester (Master) on a Connection for Acyclic Data Transfer (M-M or M-S)

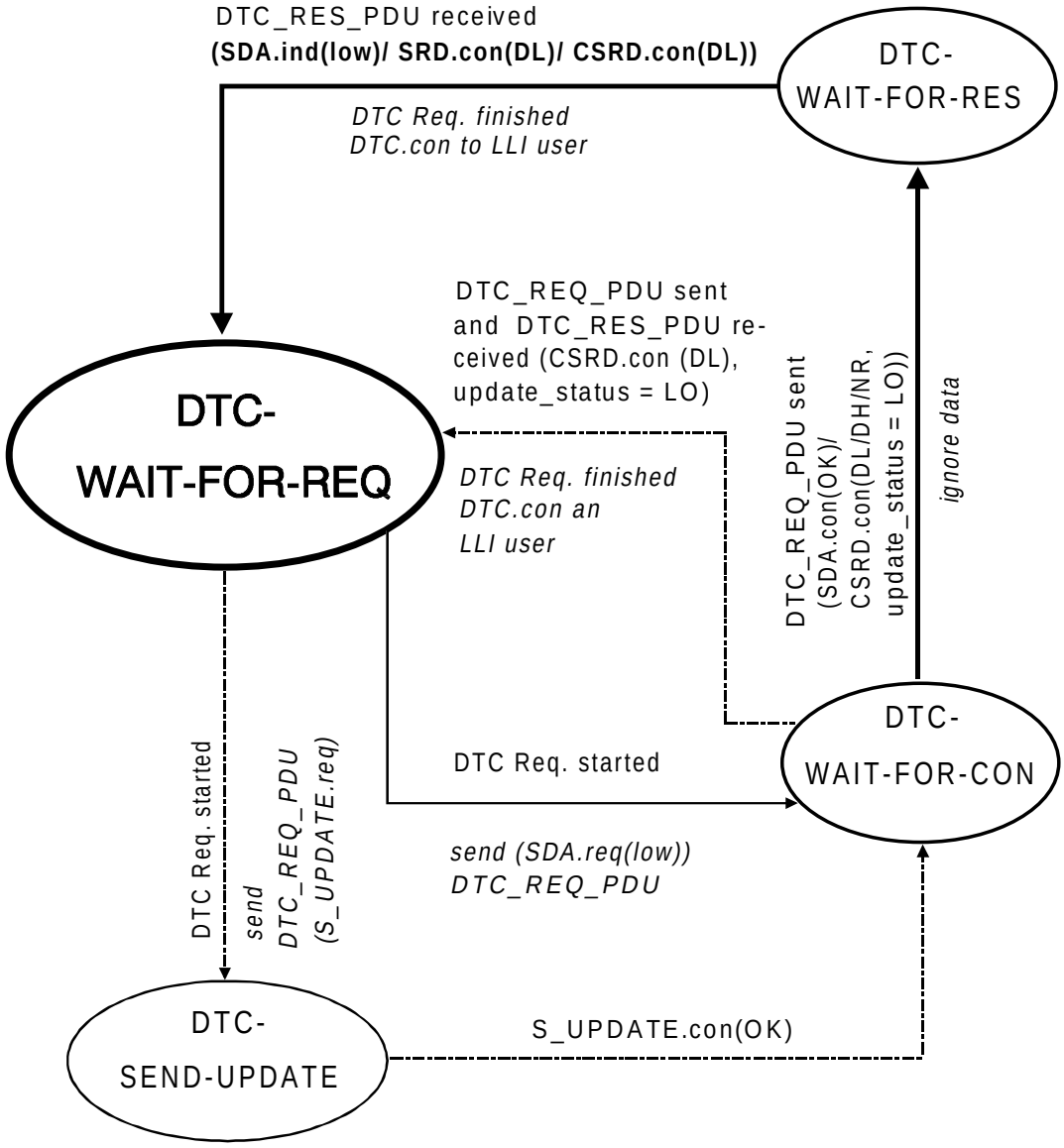


Figure 114. State Diagram for DTC Requester (Master) on a Connection for Acyclic Data Transfer (M-M or M-S)

Description of State Transitions for DTC Requester at the Master (acyc. M-M or M-S)

DTC Requester at the Master (acyc. M-M or M-S)		
Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
DTC-WAIT-FOR-REQ DTC Req. started \M-M => send DTC_REQ_PDU (SDA.req(low))	**DTC 1**	**DTC-WAIT-FOR-CON**
DTC-WAIT-FOR-REQ DTC Req. started \M-S => send DTC_REQ_PDU (S_UPDATE.req)	**DTC 2**	**DTC-SEND-UPDATE**
DTC-SEND-UPDATE update buffer loaded (S_UPDATE.con(OK)) \M-S	**DTC 3**	**DTC-WAIT-FOR-CON**
DTC-WAIT-FOR-CON DTC_REQ_PDU sent (SDA.con(OK)) \M-M	**DTC 4**	**DTC-WAIT-FOR-RES**
DTC-WAIT-FOR-CON DTC_REQ_PDU sent (CSRD.con(L_status = DL/DH/NR, update_status = LO)) \M-S AND NOT DTC_RES_PDU received (CSRD.con(L_status = DL, update_status = LO)) => ignore data if present	**DTC 5**	**DTC-WAIT-FOR-RES**
DTC-WAIT-FOR-CON DTC_REQ_PDU sent AND DTC_RES_PDU received (CSRD.con(L_status = DL, update_status = LO)) \M-S => finish DTC Req. DTC.con to LLI user	**DTC 6**	**DTC-WAIT-FOR-REQ**
DTC-WAIT-FOR-RES DTC_RES_PDU received (SDA.ind(serv_class = low)/ (C)SRD.con(L_status = DL)) \M-M OR M-S => finish DTC Req. DTC.con to LLI user	**DTC 7**	**DTC-WAIT-FOR-REQ**

6.7.3.4 State Diagram for DTC Responder (Master or Slave) on a Connection for Acyclic Data Transfer (M-M or M-S)

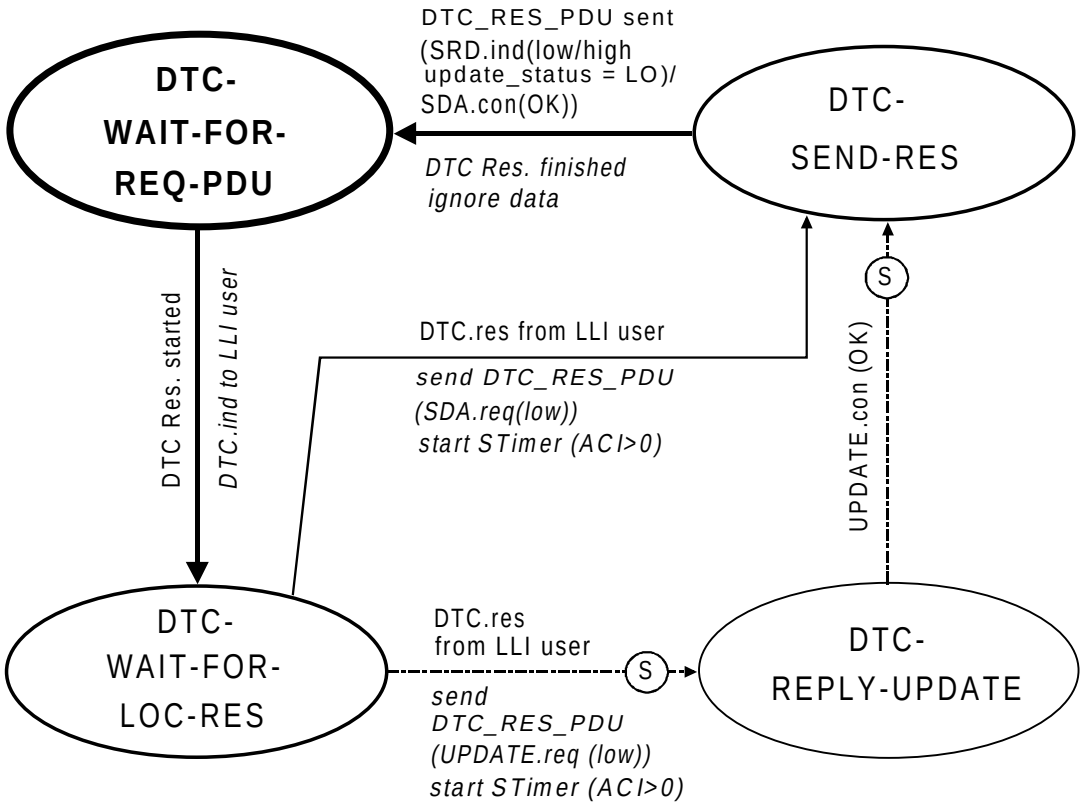


Figure 115. State Diagram for DTC Responder (Master or Slave) on a Connection for Acyclic Data Transfer (M-M or M-S)

6.7.3.5 Description of State Transitions for DTC Responder at the Master or Slave (acyc. M-M or M-S)

DTC Responder at the Master or Slave (acyc. M-M or M-S)

Current State Event \Exit Condition => Action Taken	Transition	Next State
DTC-WAIT-FOR-REQ_PDU DTC Res. started \M-M OR M-S => DTC.ind to LLI user	**DTC 1**	**DTC-WAIT-FOR-LOC-RES**
DTC-WAIT-FOR-LOC-RES DTC.res from LLI user \M-M AND ACI > 0 => send DTC_RES_PDU (SDA.req(low)) start Stimer	**DTC 2**	**DTC-SEND-RES**
DTC-WAIT-FOR-LOC-RES DTC.res from LLI user \M-M AND ACI = 0 => send DTC_RES_PDU (SDA.req(low))	**DTC 6**	**DTC-SEND-RES**
DTC-WAIT-FOR-LOC-RES DTC.res from LLI user \M-S AND ACI > 0 => send DTC_RES_PDU (UPDATE.req(low)) start STimer	**DTC 3**	**DTC-REPLY-UPDATE**
DTC-WAIT-FOR-LOC-RES DTC.res from LLI user \M-S AND ACI = 0 => send DTC_RES_PDU (UPDATE.req(low))	**DTC 7**	**DTC-REPLY-UPDATE**
DTC-REPLY-UPDATE update buffer loaded (UPDATE.con(OK)) \M-S	**DTC 4**	**DTC-SEND-RES**
DTC-SEND-RES DTC_RES_PDU sent (SDA.con(OK)/ SRD.ind(serv_class = low/high, update_status = LO)) \M-M OR M-S => finish DTC Res. ignore data if present	**DTC 5**	**DTC-WAIT-FOR-REQ_PDU**

6.7.3.6 State Diagram for IDLE Requester (Master) on a Connection for acyclic Data Transfer (M-M or M-S)

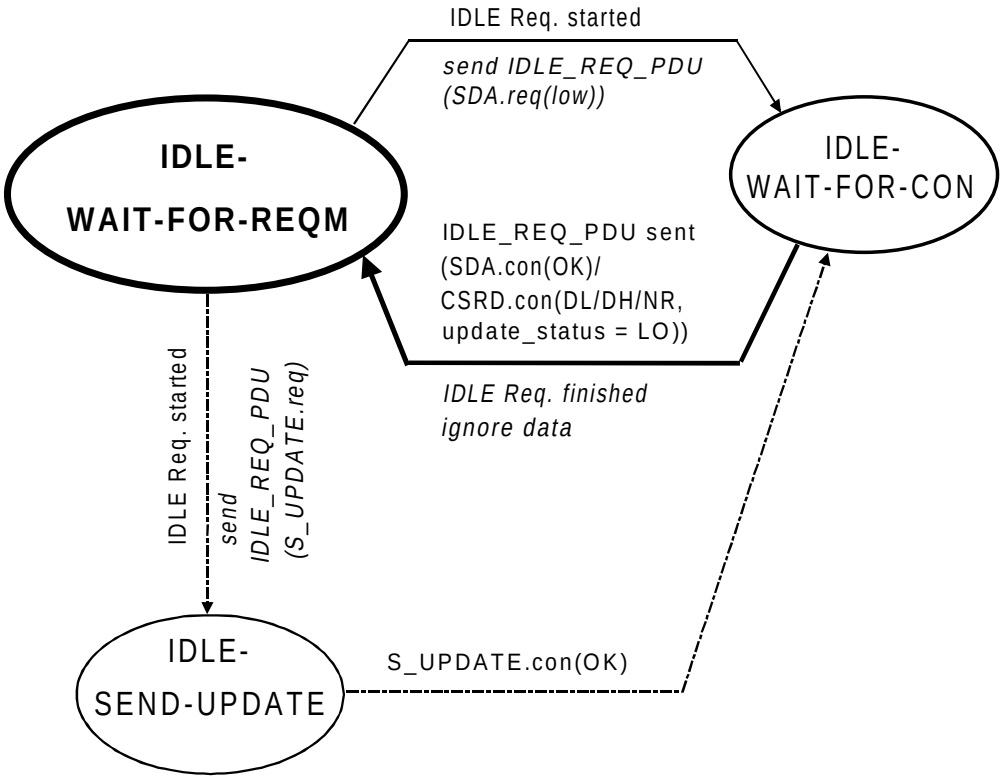


Figure 116. State Diagram for IDLE Requester at the Master

Description of State Transitions for IDLE Requester at the Master

IDLE Requester at the Master

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
IDLE-WAIT-FOR-REQM IDLE Req. started \M-M => send IDLE_REQ_PDU (SDA.req(low))	**IDLE_M 1**	**IDLE-WAIT-FOR-CON**
IDLE-WAIT-FOR-REQM IDLE Req. started \M-S => send IDLE_REQ_PDU (S_UPDATE.req)	**IDLE_M 2**	**IDLE-SEND-UPDATE**
IDLE-WAIT-FOR-CON IDLE_REQ_PDU sent (SDA.con(OK)/ CSRD.con(L_status = DL/DH/NR, update_status = LO)) \M-M OR M-S => finish IDLE Req. ignore data if present	**IDLE_M 3**	**IDLE-WAIT-FOR-REQM**
IDLE-SEND-UPDATE update buffer loaded (S_UPDATE.con(OK)) \M-S	**IDLE_M 4**	**IDLE-WAIT-FOR-CON**

6.7.3.7 State Diagram for IDLE Requester (Slave) on a Connection for Acyclic Data Transfer

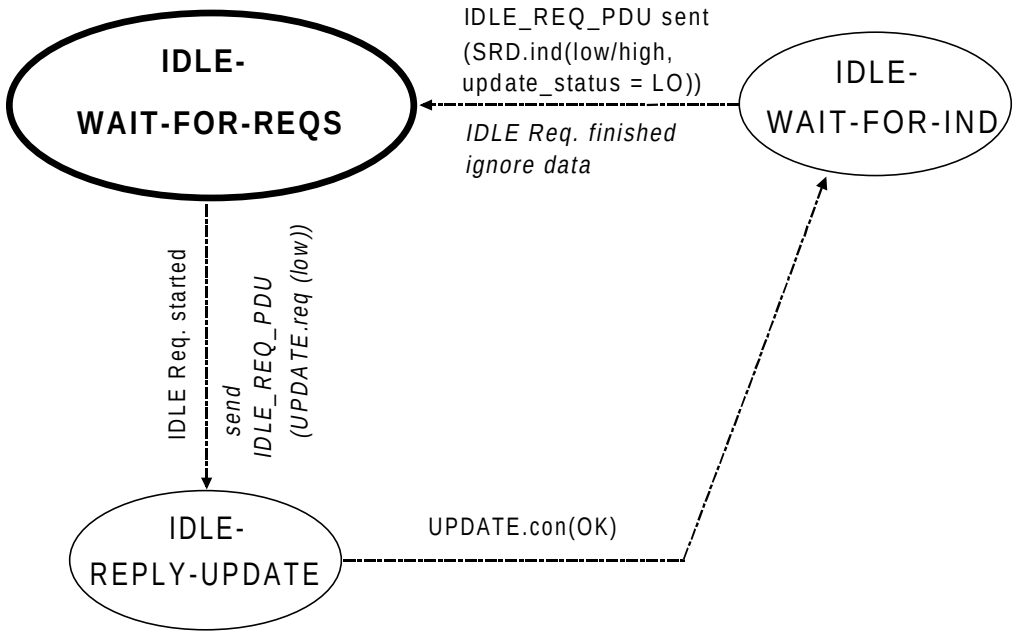


Figure 117. State Diagram for IDLE Requester at the Slave

Description of State Transitions for Idle Requester at the Slave

IDLE Requester at the Slave

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
IDLE-WAIT-FOR-REQS IDLE Req. started \M-S => send IDLE_REQ_PDU (UPDATE.req(low))	**IDLE_S 1**	**IDLE-REPLY-UPDATE**
IDLE-REPLY-UPDATE update buffer loaded (UPDATE.con(OK)) \M-S	**IDLE_S 2**	**IDLE-WAIT-FOR-IND**
IDLE-WAIT-FOR-IND IDLE_REQ_PDU sent (SRD.ind(serv_class = low/high, update_status = LO)) \M-S => finish IDLE Req. ignore data if present	**IDLE_S 3**	**IDLE-WAIT-FOR-REQS**

6.7.3.8 State Diagram for DTC Requester (Master) on a Connection for Cyclic Data Transfer

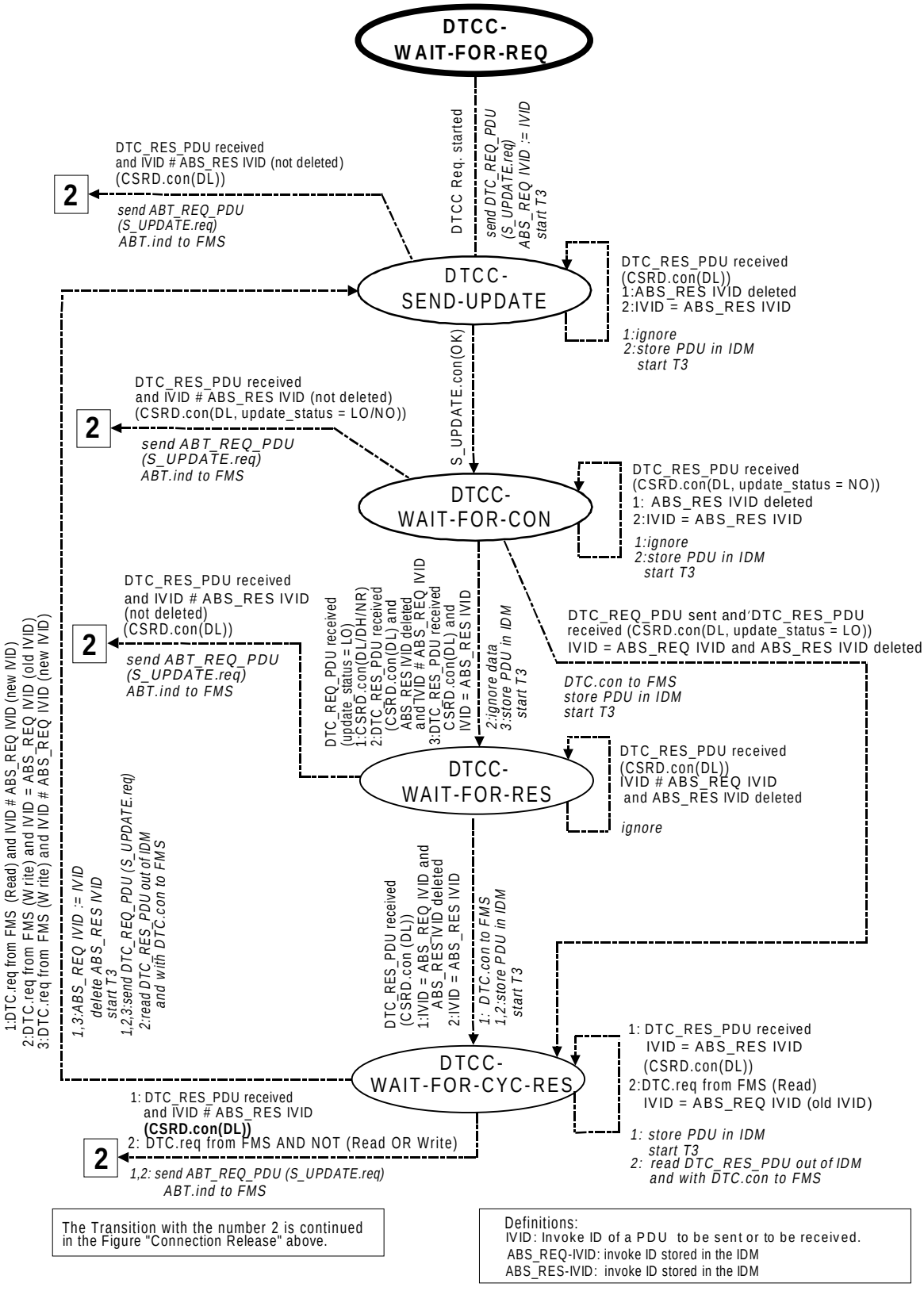


Figure 118. State Diagram for DTC Requester (Master) on a Connection for Cyclic Data Transfer

Description of State Transitions for DTC Requester at the Master (cyclic)

DTC Requester at the Master (cyclic)

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
DTCC-WAIT-FOR-REQ DTCC Req. started \M-S => send DTC_REQ_PDU (S_UPDATE.req) IDM_REQ-IVID := invoke ID start T3	**DTCC_REQ 1**	**DTCC-SEND-UPDATE**
DTCC-SEND-UPDATE update buffer loaded (S_UPDATE.con(OK)) \M-S	**DTCC_REQ 2**	**DTCC-WAIT-FOR-CON**
DTCC-SEND-UPDATE DTC_RES_PDU received ((C)SRD.con(L_status = DL)) \M-S AND (IDM_RES-IVID deleted) => ignore	**DTCC_REQ 3**	**DTCC-SEND-UPDATE**
DTCC-SEND-UPDATE DTC_RES_PDU received ((C)SRD.con(L_status = DL)) \M-S AND (invoke ID = IDM_RES-IVID) => store PDU in IDM start T3	**DTCC_REQ 4**	**DTCC-SEND-UPDATE**
DTCC-SEND-UPDATE DTC_RES_PDU received ((C)SRD.con(L_status = DL)) \M-S AND (invoke ID # IDM_RES-IVID) => send ABT_REQ_PDU <RC = ABT_RC9> (S_UPDATE.req) ABT.ind to LLI user <RC = ABT_RC22> stop all timers, start T2, stop machines	**AB 1**	**ABT-UPDATE**[1)]
DTCC-WAIT-FOR-CON DTC_REQ_PDU sent AND NOT DTC_RES_PDU received (CSRD.con(L_status = DL/DH/NR, update_status = LO)) \M-S	**DTCC_REQ 5**	**DTCC-WAIT-FOR-RES**
DTCC-WAIT-FOR-CON DTC_REQ_PDU sent AND DTC_RES_PDU received (CSRD.con(L_status = DL, update_status = LO)) \M-S AND (invoke ID = IDM_REQ-IVID) AND (IDM_RES-IVID deleted) => DTC.con to LLI user store PDU in IDM start T3	**DTCC_REQ 16**	**DTCC-WAIT-FOR-CYC-RES**
DTCC-WAIT-FOR-CON DTC_REQ_PDU sent AND DTC_RES_PDU received (CSRD.con(L_status = DL, update_status = LO)) \M-S AND (invoke ID # IDM_REQ-IVID) AND (IDM_RES-IVID deleted) => ignore data	**DTCC_REQ 17**	**DTCC-WAIT-FOR-RES**

1) see state diagram for connection release

DTC Requester at the Master (cyclic)

Current State Event \Exit Condition => Action Taken	Transition	Next State
DTCC-WAIT-FOR-CON DTC_REQ_PDU sent AND DTC_RES_PDU received (CSRD.con(L_status = DL, update_status = LO)) \M-S AND (invoke ID = IDM_RES-IVID) => store PDU in IDM start T3	**DTCC_REQ 18**	**DTCC-WAIT-FOR-RES**
DTCC-WAIT-FOR-CON DTC_RES_PDU received ((C)SRD.con(L_status = DL, update_status = NO)) \M-S AND (IDM_RES-IVID deleted) => ignore	**DTCC_REQ 6**	**DTCC-WAIT-FOR-CON**
DTCC-WAIT-FOR-CON DTC_RES_PDU received ((C)SRD.con(L_status = DL, update_status = NO)) \M-S AND (invoke ID = IDM_RES-IVID) => store PDU in IDM start T3	**DTCC_REQ 7**	**DTCC-WAIT-FOR-CON**
DTCC-WAIT-FOR-CON DTC_RES_PDU received ((C)SRD.con(L_status = DL, update_status = NO/LO)) \M-S AND (invoke ID # IDM_RES-IVID) => send ABT_REQ_PDU <RC = ABT_RC9> (S_UPDATE.req) ABT.ind to LLI user <RC = ABT_RC22> stop all timers, start T2, stop machines	**AB 2**	**ABT-UPDATE**[1)]
DTCC-WAIT-FOR-RES DTC_RES_PDU received ((C)SRD.con(L_status = DL)) \M-S AND (invoke ID = IDM_REQ-IVID) AND (IDM_RES-IVID deleted) => DTC.con to LLI user store PDU in IDM, start T3	**DTCC_REQ 8**	**DTCC-WAIT-FOR-CYC-RES**
DTCC-WAIT-FOR-RES DTC_RES_PDU received ((C)SRD.con(L_status = DL)) \M-S AND (invoke ID = IDM_RES-IVID) => store PDU in IDM start T3	**DTCC_REQ 9**	**DTCC-WAIT-FOR-CYC-RES**
DTCC-WAIT-FOR-RES DTC_RES_PDU received ((C)SRD.con(L_status = DL)) \M-S AND (invoke ID # IDM_REQ-IVID) AND (IDM_RES-IVID deleted) => ignore	**DTCC_REQ 10**	**DTCC-WAIT-FOR-RES**
DTCC-WAIT-FOR-RES DTC_RES_PDU received ((C)SRD.con(L_status = DL)) \M-S AND (invoke ID # IDM_RES-IVID) => send ABT_REQ_PDU <RC = ABT_RC22> (S_UPDATE.req) ABT.ind to LLI user <RC = ABT_RC22> stop all timers, start T2, stop machines	**AB 3**	**ABT-UPDATE**[1)]

1) see state diagram for connection release

DTC Requester at the Master (cyclic)

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
DTCC-WAIT-FOR-CYC-RES	**DTCC_REQ 11**	**DTCC-SEND-UPDATE**
DTC.req from LLI user (Read) \M-S AND (invoke ID # IDM_REQ-IVID (new IVID)) => IDM_REQ-IVID := invoke ID; delete IDM_RES-IVID, start T3; send DTC_REQ_PDU (S_UPDATE.req)		
DTCC-WAIT-FOR-CYC-RES	**DTCC_REQ 12**	**DTCC-SEND-UPDATE**
DTC.req from LLI user (Write) \M-S AND (invoke ID = IDM_REQ-IVID (old IVID)) => send DTC_REQ_PDU (S_UPDATE.req); read DTC_RES_PDU out of IDM and DTC.con to LLI user		
DTCC-WAIT-FOR-CYC-RES	**DTCC_REQ 13**	**DTCC-SEND-UPDATE**
DTC.req from LLI user (Write) \M-S AND (invoke ID # IDM_REQ-IVID (new IVID)) => IDM_REQ-IVID := invoke ID; delete IDM_RES-IVID, start T3; send DTC_REQ_PDU (S_UPDATE.req)		
DTCC-WAIT-FOR-CYC-RES	**DTCC_REQ 14**	**DTCC-WAIT-FOR-CYC-RES**
DTC_RES_PDU received ((C)SRD.con(L_status = DL)) \M-S AND (invoke ID = IDM_RES-IVID) => store PDU in IDM; start T3		
DTCC-WAIT-FOR-CYC-RES	**DTCC_REQ 15**	**DTCC-WAIT-FOR-CYC-RES**
DTC.req from LLI user (Read) \M-S AND (invoke ID = IDM_REQ-IVID (old IVID)) => read DTC_RES_PDU out of IDM and with DTC.con to LLI user		
DTCC-WAIT-FOR-CYC-RES	**AB 4**	**ABT-UPDATE[1)]**
DTC_RES_PDU received ((C)SRD.con(L_status = DL)) \M-S AND (invoke ID # IDM_RES-IVID) => send ABT_REQ_PDU <RC = ABT_RC22>, (S_UPDATE.req); ABT.ind to LLI user <RC = ABT_RC22>; stop all timers, start T2, stop machines		
DTCC-WAIT-FOR-CYC-RES	**AB 5**	**ABT-UPDATE[1)]**
DTC.req from LLI user \M-S AND NOT (Read OR Write) => send ABT_REQ_PDU <RC = ABT_RC9> (S_UPDATE.req); ABT.ind to LLI user <RC = ABT_RC27>; stop all timers, start T2, stop machines		

1) see state diagram for connection release

6.7.3.9 State Diagram for DTC Responder (Slave) ona Connection for Cyclic Data Transfer

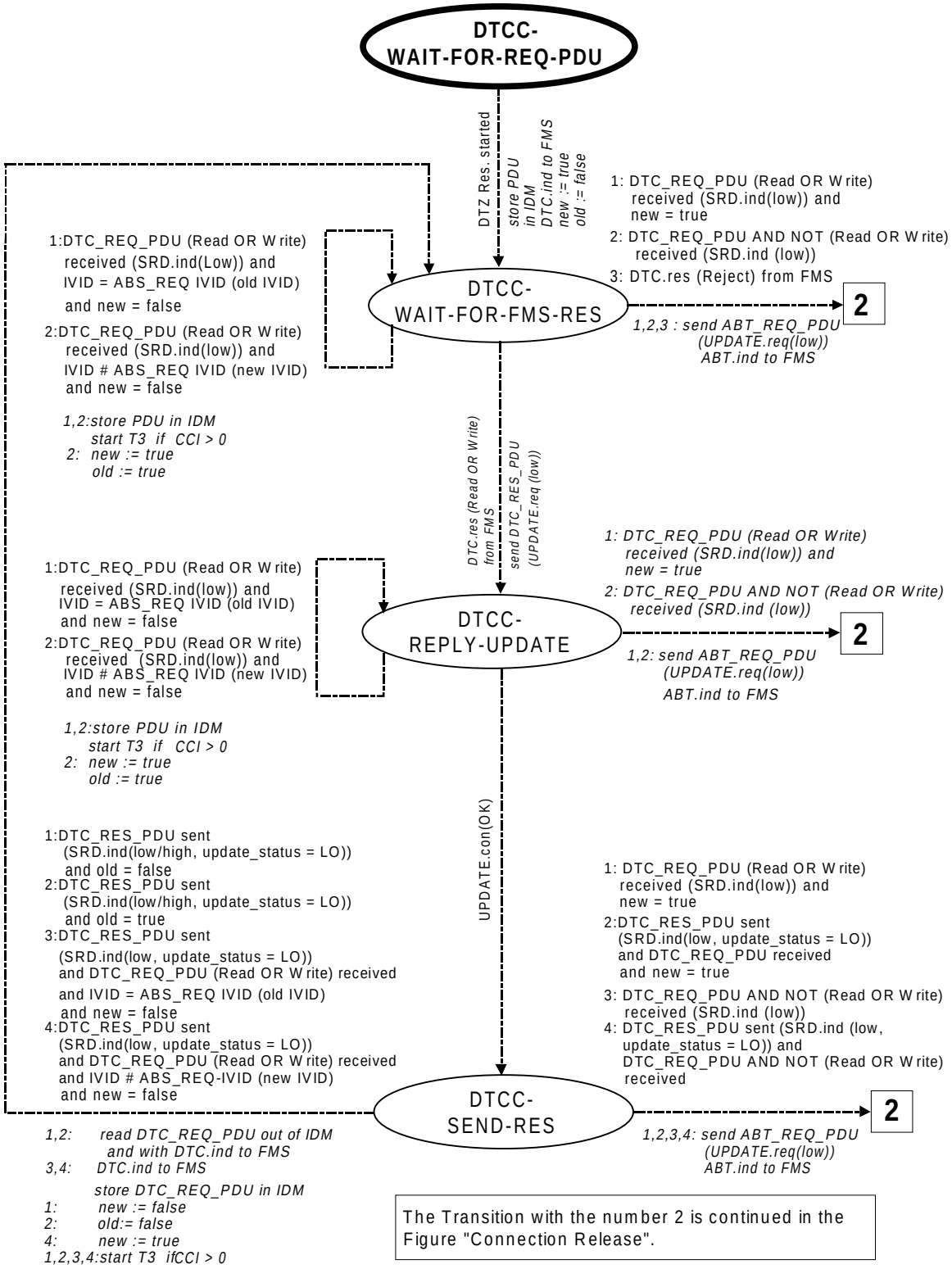


Figure 119. State Diagram for DTC Responder (Slave) on a Connection for Cyclic Data Transfer

Description of State Transitions for DTC Responder at the Slave (cyclic)

DTC Responder at the Slave (cyclic)

Current State / Event / \Exit Condition / => Action Taken	Transition	Next State
DTCC-WAIT-FOR-REQ-PDU DTCC Res. started \M-S => store PDU in IDM DTC.ind to LLI user new := true, old := false	**DTCC_RES 1**	**DTCC-WAIT-FOR-FMS-RES**
DTCC-WAIT-FOR-FMS-RES DTC.res from LLI user \M-S AND (Read OR Write) => send DTC_RES_PDU (UPDATE.req(low))	**DTCC_RES 2**	**DTCC-REPLY-UPDATE**
DTCC-WAIT-FOR-FMS-RES DTC_REQ_PDU received (SRD.ind(serv_class = low)) \M-S AND (invoke ID = IDM_REQ-IVID (old IVID)) AND (new = false) AND (CCI = 0) AND (Read OR Write) => store DTC_REQ_PDU in IDM	**DTCC_RES 3**	**DTCC-WAIT-FOR-FMS-RES**
DTCC-WAIT-FOR-FMS-RES DTC_REQ_PDU received (SRD.ind(serv_class = low)) \M-S AND (invoke ID = IDM_REQ-IVID (old IVID)) AND (new = false) AND (CCI > 0) AND (Read OR Write) => store DTC_REQ_PDU in IDM start T3	**DTCC_RES 4**	**DTCC-WAIT-FOR-FMS-RES**
DTCC-WAIT-FOR-FMS-RES DTC_REQ_PDU received (SRD.ind(serv_class = low)) \M-S AND (invoke ID # IDM_REQ-IVID (new IVID)) AND (new = false) AND (CCI = 0) AND (Read OR Write) => store DTC_REQ_PDU in IDM new := true, old := true	**DTCC_RES 5**	**DTCC-WAIT-FOR-FMS-RES**
DTCC-WAIT-FOR-FMS-RES DTC_REQ_PDU received (SRD.ind(serv_class = low)) \M-S AND (invoke ID # IDM_REQ-IVID (new IVID)) AND (new = false) AND (CCI > 0) AND (Read OR Write) => store DTC_REQ_PDU in IDM start T3 new := true, old := true	**DTCC_RES 6**	**DTCC-WAIT-FOR-FMS-RES**
DTCC-REPLY-UPDATE update buffer loaded (UPDATE.con(OK)) \M-S	**DTCC_RES 7**	**DTCC-SEND-RES**

DTC Responder at the Slave (cyclic)

Current State	Transition	Next State
Event \Exit Condition => Action Taken		
DTCC-WAIT-FOR-FMS-RES	**AB 1**	**ABT-UPDATE**[1)]
DTC_REQ_PDU received (SRD.ind(serv_class = low)) \M-S AND (new = true) => send ABT_REQ_PDU <RC = ABT_RC22> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC22> stop all timers, start T2, stop machines		
DTCC-WAIT-FOR-FMS-RES	**AB 2**	**ABT-UPDATE**[1)]
DTC_REQ_PDU received (SRD.ind(serv_class = low)) \M-S AND NOT (Read or Write) => send ABT_REQ_PDU <RC = ABT_RC27> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC27> stop all timers, start T2, stop machines		
DTCC-WAIT-FOR-FMS-RES	**AB 3**	**ABT-UPDATE**[1)]
DTC.res from LLI user \M-S AND Reject => send ABT_REQ_PDU <RC = ABT_RC28> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC28> stop all timers, start T2, stop machines		
DTCC-REPLY-UPDATE	**DTCC_RES 8**	**DTCC-REPLY-UPDATE**
DTC_REQ_PDU received (SRD.ind(serv_class = low)) \M-S AND (invoke ID = IDM_REQ-IVID (old IVID)) AND (new = false) AND (CCI = 0) AND (Read OR Write) => store DTC_REQ_PDU in IDM		
DTCC-REPLY-UPDATE	**DTCC_RES 9**	**DTCC-REPLY-UPDATE**
DTC_REQ_PDU received (SRD.ind(serv_class = low)) \M-S AND (invoke ID = IDM_REQ-IVID (old IVID)) AND (new = false) AND (CCI > 0) AND (Read OR Write) => store DTC_REQ_PDU in IDM start T3		
DTCC-REPLY-UPDATE	**DTCC_RES 10**	**DTCC-REPLY-UPDATE**
DTC_REQ_PDU received (SRD.ind(serv_class = low)) \M-S AND (invoke ID # IDM_REQ-IVID (new IVID)) AND (new = false) AND (CCI = 0) AND (Read OR Write) => store DTC_REQ_PDU in IDM new := true, old := true		

1) see state diagram for connection release

DTC Responder at the Slave (cyclic)

Current State Event \Exit Condition => Action Taken	Transition	Next State
DTCC-REPLY-UPDATE DTC_REQ_PDU received (SRD.ind(serv_class = low)) \M-S AND (invoke ID # IDM_REQ-IVID (new IVID)) AND (new = false) AND (CCI > 0) AND (Read OR Write) => store DTC_REQ_PDU in IDM start T3 new := true, old := true	**DTCC_RES 11**	**DTCC-REPLY-UPDATE**
DTCC-REPLY-UPDATE DTC_REQ_PDU received (SRD.ind(serv_class = low)) \M-S AND NOT (Read or Write) => send ABT_REQ_PDU <RC = ABT_RC27> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC27> stop all timers, start T2, stop machines	**AB 4**	**ABT-UPDATE**[1)]
DTCC-REPLY-UPDATE DTC_REQ_PDU received (SRD.ind(serv_class = low)) \M-S AND (new = true) => send ABT_REQ_PDU <RC = ABT_RC22> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC22> stop all timers, start T2, stop machines	**AB 5**	**ABT-UPDATE**[1)]
DTCC-SEND-RES DTC_RES_PDU sent (SRD.ind(serv_class = low/high, update_status = LO)) \M-S AND (old = false) AND (CCI = 0) => read DTC_REQ_PDU out of IDM and with DTC.ind to LLI user new := false	**DTCC_RES 12**	**DTCC-WAIT-FOR-FMS-RES**
DTCC-SEND-RES DTC_RES_PDU sent (SRD.ind(serv_class = low/high, update_status = LO)) \M-S AND (old = false) AND (CCI > 0) => read DTC_REQ_PDU out of IDM and with DTC.ind to LLI user new := false start T3	**DTCC_RES 13**	**DTCC-WAIT-FOR-FMS-RES**
DTCC-SEND-RES DTC_RES_PDU sent (SRD.ind(serv_class = low/high, update_status = LO)) \M-S AND (old = true) AND (CCI = 0) => read DTC_REQ_PDU out of IDM and with DTC.ind to LLI user old := false	**DTCC_RES 14**	**DTCC-WAIT-FOR-FMS-RES**

1) see state diagram for connection release

DTC Responder at the Slave (cyclic)

Current State Event \Exit Condition => Action Taken	Transition	Next State
DTCC-SEND-RES DTC_RES_PDU sent (SRD.ind(serv_class = low/high, update_status = LO)) \M-S AND (old = true) AND (CCI > 0) => read DTC_REQ_PDU out of IDM and with DTC.ind to LLI user old := false start T3	**DTCC_RES 15**	**DTCC-WAIT-FOR-FMS-RES**
DTCC-SEND-RES DTC_RES_PDU sent (SRD.ind(serv_class = low, update_status = LO)) \M-S AND (DTC_REQ_PDU received AND invoke ID = IDM_REQ-IVID (old IVID)) AND (new = false) AND (CCI = 0) AND (Read OR Write) => DTC.ind to LLI user store DTC_REQ_PDU in IDM	**DTCC_RES 16**	**DTCC-WAIT-FOR-FMS-RES**
DTCC-SEND-RES DTC_RES_PDU sent (SRD.ind(serv_class = low, update_status = LO)) \M-S AND (DTC_REQ_PDU received AND invoke ID = IDM_REQ-IVID (old IVID)) AND (new = false) AND (CCI > 0) AND (Read OR Write) => DTC.ind to LLI user store DTC_REQ_PDU in IDM start T3	**DTCC_RES 17**	**DTCC-WAIT-FOR-FMS-RES**
DTCC-SEND-RES DTC_RES_PDU sent (SRD.ind(serv_class = low, update_status = LO)) \M-S AND (DTC_REQ_PDU received AND invoke ID # IDM_REQ-IVID (new IVID)) AND (new = false) AND (CCI = 0) AND (Read OR Write) => DTC.ind to LLI user store DTC_REQ_PDU in IDM, new := true	**DTCC_RES 18**	**DTCC-WAIT-FOR-FMS-RES**
DTCC-SEND-RES DTC_RES_PDU sent (SRD.ind(serv_class = low, update_status = LO)) \M-S AND (DTC_REQ_PDU received AND invoke ID # IDM_REQ-IVID (new IVID)) AND (new = false) AND (CCI > 0) AND (Read OR Write) => DTC.ind to LLI user store DTC_REQ_PDU in IDM, new := true start T3	**DTCC_RES 19**	**DTCC-WAIT-FOR-FMS-RES**

DTC Responder at the Slave (cyclic)

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
DTCC-SEND-RES DTC_REQ_PDU received (SRD.ind (Serv.class = low)) AND NOT (Read OR Write) \M-S => send ABT_REQ_PDU <RC = ABT_RC27> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC27> stop all timers, start T2, stop machines	**AB 6**	**ABT-UPDATE**[1)]
DTCC-SEND-RES DTC_REQ_PDU received (SRD.ind(serv_class = low)) \M-S AND (new = true) => send ABT_REQ_PDU <RC = ABT_RC22> (UPDATE.req(low)) ABT.ind to LLI user <RC = ABT_RC22> stop all timers, start T2, stop machines	**AB 7**	**ABT-UPDATE**[1)]

1) see state diagram for connection release

6.7.3.10 State Diagram for DTA Requester at the Master

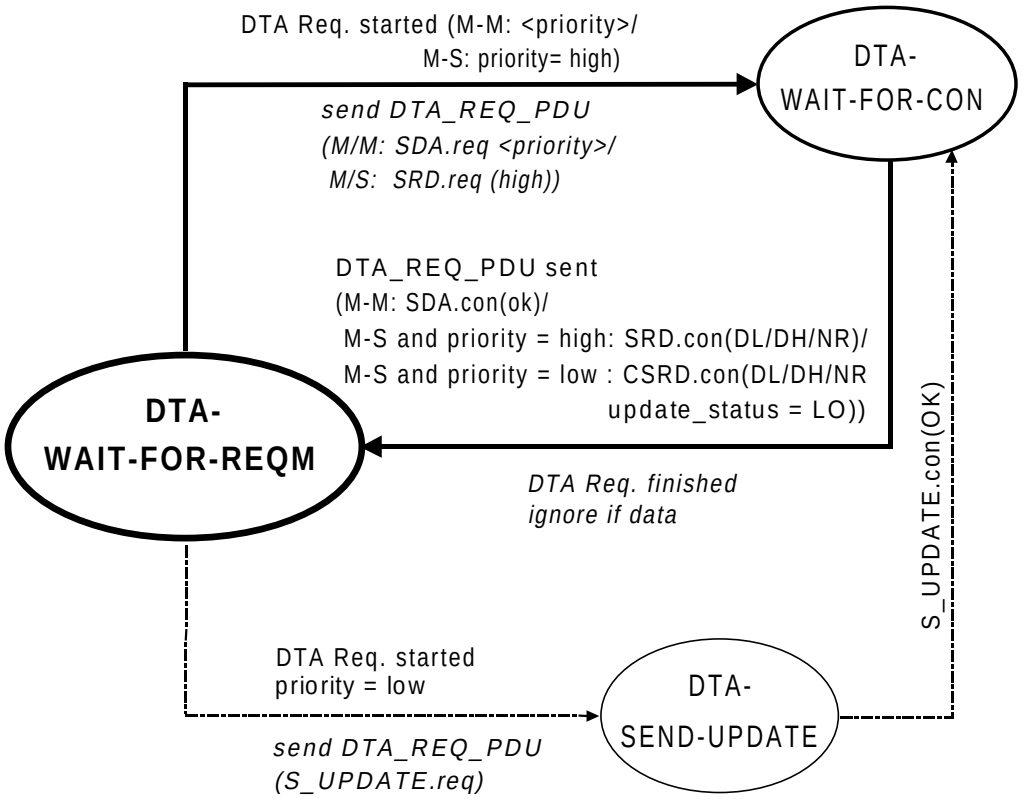


Figure 120. State Diagram for DTA Requester at the Master

Description of State Transitions for DTA Requester at the Master

DTA Requester at the Master

Current State Event \Exit Condition => Action Taken	Transition	Next State
DTA-WAIT-FOR-REQM DTA Req. started \M-S AND (priority = low) => send DTA_REQ_PDU (S_UPDATE.req)	**DTA_REQM 1**	**DTA-SEND-UPDATE**
DTA-WAIT-FOR-REQM DTA Req. started \M-S AND (priority = high) => send DTA_REQ_PDU (SRD.req(high))	**DTA_REQM 2**	**DTA-WAIT-FOR-CON**
DTA-WAIT-FOR-REQM DTA Req. started \M-M => send DTA_REQ_PDU (SDA.req(low/high))	**DTA_REQM 3**	**DTA-WAIT-FOR-CON**
DTA-SEND-UPDATE update buffer loaded (S_UPDATE.con(OK)) \M-S	**DTA_REQM 4**	**DTA-WAIT-FOR-CON**
DTA-WAIT-FOR-CON DTA_REQ_PDU sent (SRD.con(DL/DH/NR)) \M-S AND (priority = high) => finish DTA Req. ignore data if present	**DTA_REQM 5**	**DTA-WAIT-FOR-REQM**
DTA-WAIT-FOR-CON DTA_REQ_PDU sent (CSRD.con(L_status = DL/DH/NR, update_status = LO)) \M-S AND (priority = low) => finish DTA Req. ignore data if present	**DTA_REQM 6**	**DTA-WAIT-FOR-REQM**
DTA-WAIT-FOR-CON DTA_REQ_PDU sent (SDA.con(OK)) \M-M => finish DTA Req.	**DTA_REQM 7**	**DTA-WAIT-FOR-REQM**

6.7.3.11 State Diagram for DTA Requester at the Slave

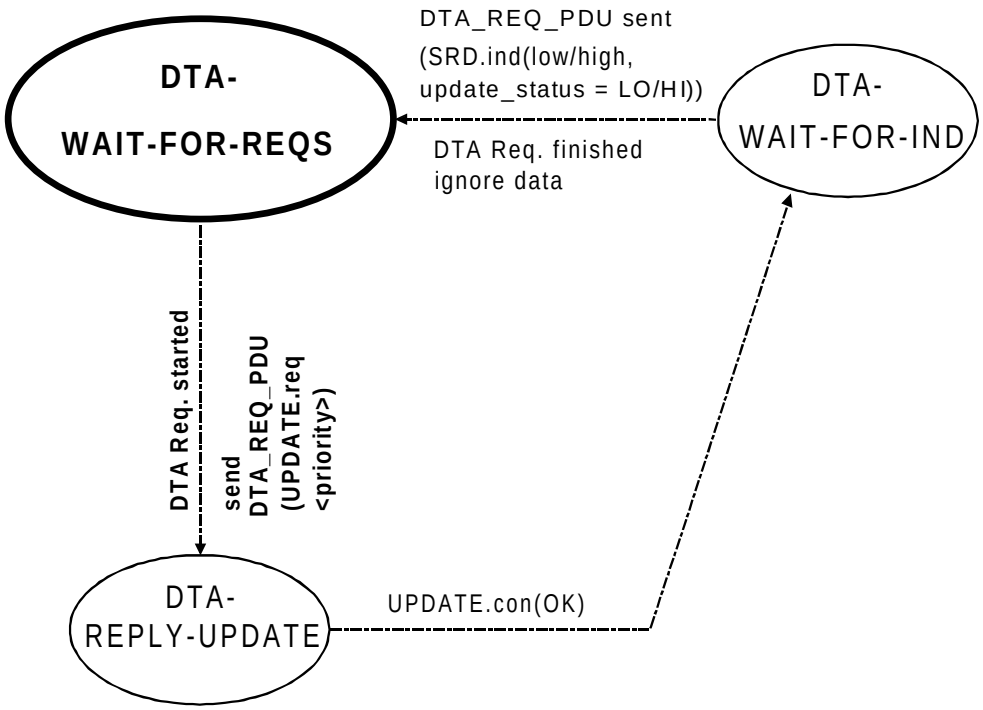


Figure 121. State Diagram for DTA Requester at the Slave

Description of State Transitions for DTA Requester at the Slave

DTA Requester at the Slave

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
DTA-WAIT-FOR-REQS DTA Req. started \M-S => send DTA_REQ_PDU (UPDATE.req(priority))	**DTA_REQS 1**	**DTA-REPLY-UPDATE**
DTA-REPLY-UPDATE update buffer loaded (UPDATE.con(OK)) \M-S	**DTA_REQS 2**	**DTA-WAIT-FOR-IND**
DTA-WAIT-FOR-IND DTA_REQ_PDU sent (SRD.ind(serv_class = low/high, update_status = LO/HI)) \M-S => finish DTA Req. ignore data if present	**DTA_REQS 3**	**DTA-WAIT-FOR-REQS**

6.7.3.12 State Diagram for DTA Acknowledge

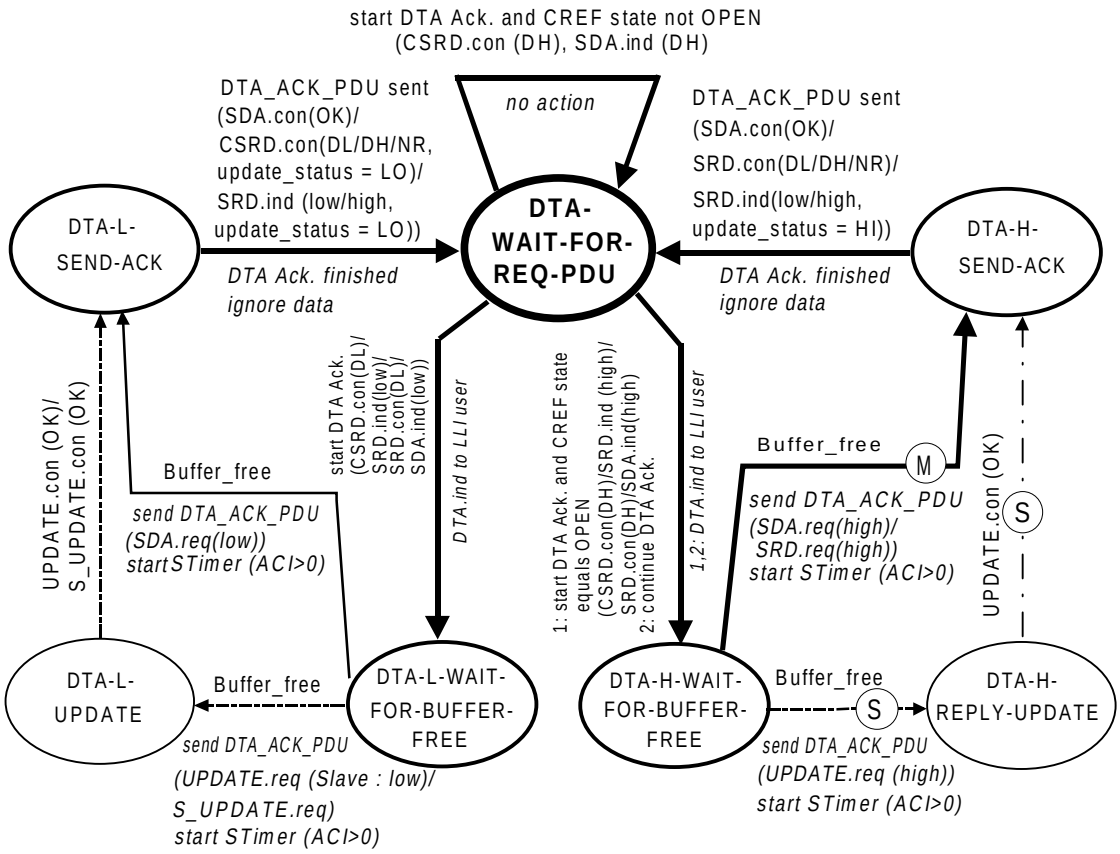


Figure 122. State Diagram for DTA Acknowledge

Description of State Transitions for DTA Acknowledge

DTA Acknowledge

Current State / Event / \Exit Condition / => Action Taken	Transition	Next State
DTA-WAIT-FOR-REQ-PDU DTA Ack. started (CSRD.con(L_status = DH)/ SRD.ind(serv_class = high)/SRD.con(DH)/ SDA.ind(serv_class = high)) \(M-M OR M-S) AND CREF state = OPEN => DTA.ind to LLI user	**DTA 1**	**DTA-H-WAIT-FOR-BUFFER-FREE**
DTA-WAIT-FOR-REQ-PDU DTA Ack. started (CSRD.con(L_status = DL)/ SRD.ind(serv_class = low)/SRD.con(DL)/ SDA.ind(serv_class = low)) \(M-M OR M-S) AND CREF state = OPEN => DTA.ind to LLI user	**DTA 2**	**DTA-L-WAIT-FOR-BUFFER-FREE**
DTA-WAIT-FOR-REQ-PDU DTA Ack. started (CSRD.con(L_status = DH)/ SDA.ind(serv_class = high)) \(M-M OR M-S) AND CREF state # OPEN => no action	**DTA 12**	**DTA-WAIT-FOR-REQ-PDU**
DTA-WAIT-FOR-REQ-PDU continue DTA-Ack. \(M-M OR M-S with SI) => DTA.ind to LLI user	**DTA 13**	**DTA-H-WAIT-FOR-BUFFER-FREE**
DTA-H-WAIT-FOR-BUFFER-FREE free memory available \(M-M OR ((acyc. M-S) AND Master)) AND ACI > 0 => send DTA_ACK_PDU (SDA.req(high)/SRD.req(high)) start Stimer	**DTA 3**	**DTA-H-SEND-ACK**
DTA-H-WAIT-FOR-BUFFER-FREE free memory available \(M-M OR ((acyc. M-S) AND Master)) AND ACI = 0 => send DTA_ACK_PDU (SDA.req(high)/SRD.req(high))	**DTA 16**	**DTA-H-SEND-ACK**
DTA-H-WAIT-FOR-BUFFER-FREE free memory available \cyc. M-S and MASTER => send DTA_ACK_PDU (SRD.req(high))	**DTA 19**	**DTA-H-SEND-ACK**
DTA-H-WAIT-FOR-BUFFER-FREE free memory available \acyc. M-S AND Slave AND ACI > 0 => send DTA_ACK_PDU (UPDATE.req(high)) start Stimer	**DTA 4**	**DTA-H-REPLY-UPDATE**
DTA-H-WAIT-FOR-BUFFER-FREE free memory available \acyc. M-S AND Slave AND ACI = 0 => send DTA_ACK_PDU (UPDATE.req(high))	**DTA 17**	**DTA-H-REPLY-UPDATE**

DTA Acknowledge

Current State Event \Exit Condition => Action Taken	Transition	Next State
DTA-H-WAIT-FOR-BUFFER-FREE free memory available \cyc. M-S AND Slave => send DTA_ACK_PDU (UPDATE.req(high))	**DTA 20**	**DTA-H-REPLY-UPDATE**
DTA-H-REPLY-UPDATE update buffer loaded (UPDATE.con(OK)) \M-S AND Slave	**DTA 5**	**DTA-H-SEND-ACK**
DTA-H-SEND-ACK DTA_ACK_PDU sent (SDA.con(OK)/SRD.con(NR/DL/DH)/ SRD.ind(serv_class = low/high, update_status = HI)) \M-M OR ((acyc. M-S) OR(cyc. M-S AND MASTER)) => finish DTA Ack. ignore data if present	**DTA 6**	**DTA-WAIT-FOR-REQ-PDU**
DTA-H-SEND-ACK DTA_ACK_PDU sent SRD.ind(serv_class = low/high, update_status = HI)) \cyc. M-S AND SLAVE AND (CCI=0 OR status DTCC=DTCC-WAIT-FOR-REQ-PDU) => finish DTA Ack. ignore data if present	**DTA 22**	**DTA-WAIT-FOR-REQ-PDU**
DTA-H-SEND-ACK DTA_ACK_PDU sent SRD.ind(serv_class = low/high, update_status = HI)) \cyc. M-S AND SLAVE AND (CCI>0 AND status DTCC # DTCC-WAIT-FOR-REQ-PDU) => finish DTA Ack. ignore data if present start T3	**DTA 23**	**DTA-WAIT-FOR-REQ-PDU**
DTA-L-WAIT-FOR-BUFFER-FREE free memory available \M-M AND ACI > 0 => send DTA_ACK_PDU (SDA.req(low)) start Stimer	**DTA 7**	**DTA-L-SEND-ACK**
DTA-L-WAIT-FOR-BUFFER-FREE free memory available \M-M AND ACI = 0 => send DTA_ACK_PDU (SDA.req(low))	**DTA 18**	**DTA-L-SEND-ACK**
DTA-L-WAIT-FOR-BUFFER-FREE free memory available \acyc. M-S AND ACI>0 => send DTA_ACK_PDU (S_UPDATE.req/UPDATE.req(low)) start STimer	**DTA 8**	**DTA-L-UPDATE**
DTA-L-WAIT-FOR-BUFFER-FREE free memory available \(acyc. M-S AND ACI=0) OR (cyc. M-S) => send DTA_ACK_PDU (S_UPDATE.req/UPDATE.req(low))	**DTA 21**	**DTA-L-UPDATE**

DTA Acknowledge

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
DTA-L-UPDATE update buffer loaded (S_UPDATE.con(OK)/UPDATE.con(OK)) \M-S	**DTA 9**	**DTA-L-SEND-ACK**
DTA-L-SEND-ACK DTA_ACK_PDU sent (SDA.con(OK)/ CSRD.con(L_status = NR/DL/DH, update_status = LO)/ SRD.ind(serv_class = low/high, update_status = LO)) \M-M OR((acyc. M-S) OR(cyc. M-S AND MASTER)) => finish DTA Ack. ignore data if present	**DTA 10**	**DTA-WAIT-FOR-REQ-PDU**
DTA-L-SEND-ACK DTA_ACK_PDU sent SRD.ind(serv_class = low/high, update_status = LO)) \cyc. M-S AND SLAVE AND (CCI=0 OR status DTCC = DTCC-WAIT-FOR-REQ-PDU) => finish DTA Ack. ignore data if present	**DTA 24**	**DTA-WAIT-FOR-REQ-PDU**
DTA-L-SEND-ACK DTA_ACK_PDU sent SRD.ind(serv_class = low/high, update_status = LO)) \cyc. M-S AND SLAVE AND (CCI>0 AND status DTCC # DTCC-WAIT-FOR-REQ-PDU) => finish DTA Ack. ignore data if present start T3	**DTA 25**	**DTA-WAIT-FOR-REQ-PDU**

6.7.4 Broadcast and Multicast

6.7.4.1 State Diagram for DTU Requester (Master)

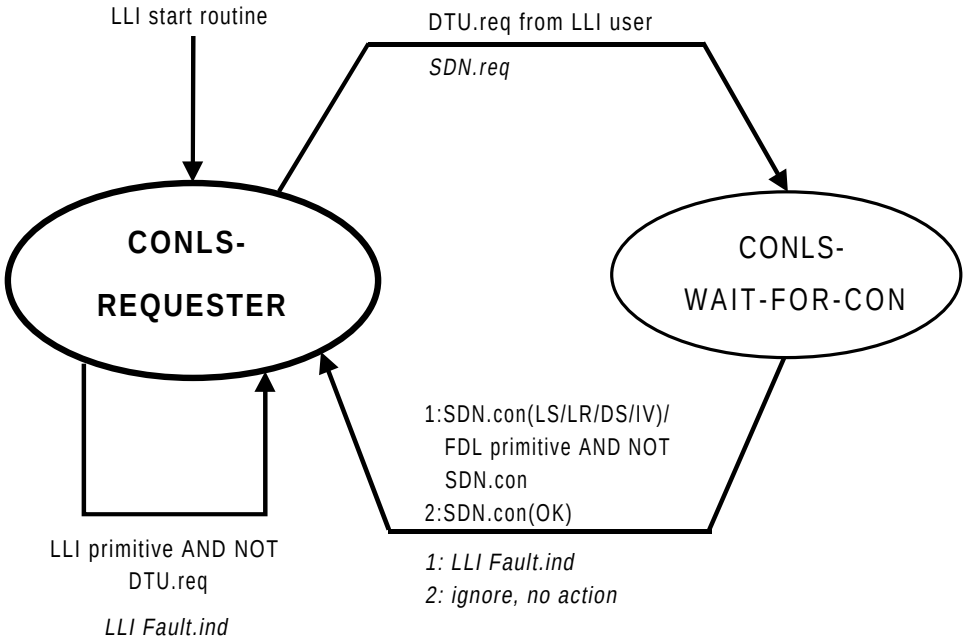


Figure 123. State Diagram for DTU Requester (Master)

Description of State Transitions for DTU Requester at the Master

DTU Requester at the Master

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
CONLS-REQUESTER LLI primitive AND NOT DTU.req \CONLS => LLI-Fault.ind <RC = LLI_FMA7_RC11, AD = code of the primitive>	**CONLS_RQ 1**	**CONLS-REQUESTER**
CONLS-REQUESTER DTU.req from LLI user \CONLS => SDN.req	**CONLS_RQ 2**	**CONLS-WAIT-FOR-CON**
CONLS-WAIT-FOR-CON SDN.con(LS/LR/DS/IV) \CONLS => LLI-Fault.ind <RC = LLI_FMA7_RC15, AD = LS/LR/DS/IV>	**CONLS_RQ 3**	**CONLS-REQUESTER**
CONLS-WAIT-FOR-CON FDL primitive AND NOT SDN.con \CONLS => LLI-Fault.ind <RC = LLI_FMA7_RC7, AD = code of the primitive>	**CONLS_RQ 4**	**CONLS-REQUESTER**
CONLS-WAIT-FOR-CON SDN.con(OK) \CONLS	**CONLS_RQ 5**	**CONLS-REQUESTER**

6.7.4.2 State Diagram for DTU Receiver (Master or Slave)

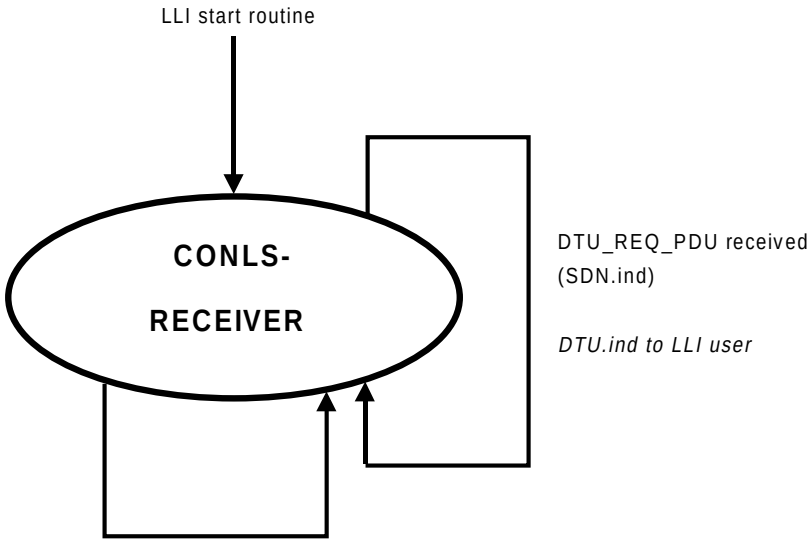


Figure 124. State Diagram for DTU Receiver (Master or Slave)

Description of State Transitions for DTU Receiver at the Master or Slave

DTU Receiver at the Master or Slave		
Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
CONLS-RECEIVER DTU_REQ_PDU received (SDN.ind) \CONLS => DTU.ind to LLI user	**CONLS_SE 1**	**CONLS-RECEIVER**
CONLS-RECEIVER FDL primitive AND NOT SDN.ind \CONLS => LLI-Fault.ind <RC = LLI_FMA7_RC7, AD = code of the primitive>	**CONLS_SE 2**	**CONLS-RECEIVER**
CONLS-RECEIVER SDN.ind AND NOT DTU_REQ_PDU \CONLS => ignore data	**CONLS_SE 3**	**CONLS-RECEIVER**

This page is intentionally left blank.

6.7.5 Overview of Layer 2 Services, Primitives and PDUs used by LLI

Depending on the types of communication relationships and LLI services which are supported by the LLI, the implementation of the related Layer 2 services and their primitives is mandatory in the Layer 2 protocol. These mandatory services and primitives are specified in the following tables.

The FMA1/2 services used by LLI do not depend on the types of communication relationships and LLI services supported. The implementation of these FMA1/2 services is mandatory in FMA1/2.

Table 24. Send Data with No Acknowledge (SDN)

Slave	Master
service primitive	service primitive
	FDL_DATA.req (SDN.req)
	FDL_DATA.con (SDN.con)
FDL_DATA.ind (SDN.ind)	FDL_DATA.ind (SDN.ind)
PDU: [SDN_REQ_PDU]	

Table 25. Send Data with Acknowledge (SDA)

Slave	Master
service primitive	service primitive
	FDL_DATA_ACK.req (SDA.req)
	FDL_DATA_ACK.con (SDA.con)
	FDL_DATA_ACK.ind (SDA.ind)
PDUs: [SDA_REQ_PDU]	[SDA _ACK_PDU]

In the formal descriptions of the LLI state machines, the abbreviations shown in round brackets are used for the service primitives.

Table 26. Send and Request Data (SRD)

Slave	Master
service primitive	service primitive
	FDL_DATA_REPLY.req (SRD.req)
	FDL_DATA_REPLY.con (SRD.con)
FDL_DATA_REPLY.ind (SRD.ind)	
FDL_REPLY_UPDATE.req (UPDATE.req)	
FDL_REPLY_UPDATE.con (UPDATE.con)	
PDUs: [SRD_REQ_PDU]	[SRD_RES_PDU]

Table 27. Cyclic Send and Request Data (CSRD)

Slave	Master
service primitive	service primitive
	FDL_SEND_UPDATE.req (S_UPDATE.req)
	FDL_SEND_UPDATE.con (S_UPDATE.con)
	FDL_CYC_DATA_REPLY.req (CSRD.req)
	FDL_CYC_DATA_REPLY.con (CSRD.con)
	FDL_CYC_ENTRY.req (ENTRY.req)
	FDL_CYC_ENTRY.con (ENTRY.con)
	FDL_CYC_DEACT.req (DEACT.req)
	FDL_CYC_DEACT.con (DEACT.con)
FDL_DATA_REPLY.ind (SRD.ind)	
FDL_REPLY_UPDATE.req (UPDATE.req)	
FDL_REPLY_UPDATE.con (UPDATE.con)	
PDUs: [SRD_REQ_PDU]	[SRD_RES_PDU]

Table 28. (R)SAP Activate/SAP Deactivate FMA1/2

Slave/Master
service primitive
FMA1/2_SAP_ACTIVATE.req (SAP_ACT.req)
FMA1/2_SAP_ACTIVATE.con (SAP_ACT.con)
FMA1/2_RSAP_ACTIVATE.req (RSAP_ACT.req)
FMA1/2_RSAP_ACTIVATE.con (RSAP_ACT.con)
FMA1/2_SAP_DEACTIVATE.req (SAP_DEACT.req)
FMA1/2_SAP_DEACTIVATE.con (SAP_DEACT.con)

6.7.6 Mapping of all LLI Services onto Layer 2 Services as a function of Communication Relationships

Legend for the following four tables:

c : service confirmed
a : service acknowledged by LLI
u : service unconfirmed
L : priority = Low
H : priority = High

The slave is not able to distinct between the CSRD and SRD services.

Table 29. Services on Master-Slave Connections with Slave Initiative

Services	c/a/u		Slave Requester	Slave Responder	Master Requester	Master Responder
DTC	c		--	(C)SRD	CSRD	--
DTA	a	L	(C)SRD	(C)SRD	CSRD	CSRD
	a	H	(C)SRD	SRD	SRD	CSRD
ASS	c		--	(C)SRD	CSRD	--
ABT	u		(C)SRD	(C)SRD	CSRD	CSRD

Table 30. Services on Master-Slave Connections with no Slave Initiative

Services	c/a/u		Slave Requester	Slave Responder	Master Requester	Master Responder
DTC	c		--	(C)SRD	CSRD	--
DTA	a	L	--	(C)SRD	CSRD	--
	a	H	--	SRD	SRD	--
ASS	c		--	(C)SRD	CSRD	--
ABT	u		(C)SRD	(C)SRD	CSRD	CSRD

Table 31. Services on Master-Master Connections

Services	c/a/u	Master Requester	Master Responder
DTC	c	SDA	SDA
DTA L	a	SDA	SDA
DTA H	a	SDA	SDA
ASS	c	SDA	SDA
ABT	u	SDA	SDA

Table 32. Services on connectionless Communication Relationships

Service	c/a/u	Slave Requester	Slave Receiver	Master Requester	Master Receiver
DTU L	u	---	SDN	SDN	SDN
DTU H	u	---	SDN	SDN	SDN

6.7.7 Abort Reason Codes

This clause contains definitions for the meaning and generation of the parameters identifier (ID), locally generated (LG), reason code (RC) and additional detail (AD) for the LLI abort service (ABT).

6.7.7.1 Locally initiated Connection Abort

The reason for a locally generated connection abort may arise at the local user, the local FMS or FMA7, the local LLI or the local Layer 2. If the local LLI initiates the connection abort (failure at the local LLI or the local Layer 2), the parameter LG of the ABT.ind, which is issued to the local user, shall be true.

If the connection abort is initiated by the local user (parameter ID = user), or from the local FMS or FMA7 (parameter ID = LLI user), the value of the parameter RC, which is delivered from the LLI user to LLI, shall be encoded by LLI (see section 4.5) and entered into the ABT_REQ_PDU. If in addition the LLI user delivers the parameter AD to LLI, then LLI shall enter this parameter into the ABT_REQ_PDU.

If the reason for the connection abort is located in LLI (parameter ID = LLI), the field reason code may take the values defined in Table 110. Depending on the reason code, the field AD may contain additional information.

If the reason for the connection abort is due to a failure in the local Layer 2, the parameter ID shall contain the value "Layer 2". In this case the parameter reason code of the ABT.ind may take the following values:

RR, LR, NA, RDL, RDH, LS, IV, OK, NO, DS, RS, UE.

The meaning of these reason codes is defined in the PROFIBUS Data Link Layer. The parameter AD shall contain additional information about the service which failed. The parameter AD may take the following values:

Table 33. Parameters AD (local)

Code	Meaning
ABT_AD1	Error in the loading of the UPDATE buffer (S_UPDATE.con/UPDATE.con)
ABT_AD2	Error in the activation of a Poll List entry (Entry.con)
ABT_AD3	Error in the deactivation of a Poll List entry (Entry.con)
ABT_AD4	Error on transmission (SDA.con)
ABT_AD5	Error on transmission (CSRD.con)
ABT_AD6	Error on transmission (SRD.con)
ABT_AD7	Error on receipt (CSRD.con)

6.7.7.2 Remotely initiated Connection Abort

A remotely initiated connection abort may be generated by the remote user (parameter ID = user), the remote FMS or FMA7 (parameter ID = LLI user), the remote LLI (parameter ID = LLI) or the remote Layer 2 (parameter ID = Layer 2). In these cases the local LLI receives an ABT_REQ_PDU from the remote LLI. Then the local LLI shall assign the value "false" to the parameter LG and enter it into the ABT.ind to be issued to the local LLI user.

If the connection abort is initiated by the remote user or remote LLI user, the LLI shall decode the value of the parameter RC (see section 4.5), which is contained in the ABT_REQ_PDU. The LLI shall transfer this value, together with the parameters ID and AD which are also contained in the ABT_REQ_PDU, to the local LLI user.

If the connection abort is initiated by the remote LLI, the values for the parameters RC and AD are defined as in the table below.

Is the reason for the connection abort is due to a failure in the remote Layer 2, the parameter reason code (RC) may take the following values:

RR, LR, NA, RDL, RDH.

The meaning of these reason codes is defined in the PROFIBUS Data Link Layer. In this case the field AD shall be omitted.

6.7.7.3 Reason Codes of LLI for the Abort.indication

Table 34. Abort Reason Codes (local and remote) for ABT.ind with Identifier = LLI

Code	Meaning	LG = false / LG = true	
ABT_RC1	LLI-LLI context check negative AD contains the remote LLI context	x	
ABT_RC2	Unallowed LLI PDU received in the connection establishment phase or release phase	x	x
ABT_RC3	Unallowed LLI PDU received in the data transfer phase	x	x
ABT_RC4	Unknown or faulty LLI PDU received	x	x
ABT_RC5	DTA_ACK_PDU received and SAC = 0	x	x
ABT_RC6	Number of parallel services exceeded (LLI PDU received)	x	x
ABT_RC7	Invoke ID unknown	x	x
ABT_RC8	Priority error	x	x
ABT_RC9	Local error in the remote Station	x	x
ABT_RC10	Timer T1 expired (connection establishment)	x	x
ABT_RC11	Timer T3 expired (connection monitoring)	x	x
ABT_RC12	RTimer expired	x	x
ABT_RC13	Error in the LSAP activation (.con(-)) AD contains the error status	x	
ABT_RC14	Unallowed FDL primitive in the connection establishment phase or release phase AD contains the code of the primitive	x	
ABT_RC15	Unallowed FDL primitive in the data transfer phase AD contains the code of the primitive	x	
ABT_RC16	Unknown FDL primitive	x	
ABT_RC17	Unknown LLI primitive	x	
ABT_RC18	Unallowed LLI primitive in the connection establishment phase or release phase AD contains the code of the primitive	x	
ABT_RC19	Unallowed LLI primitive in the data transfer phase AD contains the code of the primitive	x	
ABT_RC20	CRL entry not OK	x	
ABT_RC21	Conflict case in the connection establishment phase: local address > remote address	x	
ABT_RC22	Execution error in cyclic data transfer	x	x
ABT_RC23	Number of parallel services exceeded (Request from FMS)	x	
ABT_RC24	CRL is being loaded by FMA7, LLI disabled	x	x
ABT_RC25	Confirm / indication mode error	x	
ABT_RC26	Unallowed FMA1/2 primitive received	x	
ABT_RC27	Unallowed FMS service on connection for cyclic data transfer	x	x
ABT_RC28	FMS PDU size exceeded on connection for cyclic data transfer	x	x

6.7.8 Reason Codes for the LLI-Fault.indication

Table 35. Reason Codes for LLI-Fault.ind

Code	Meaning
LLI_FMA7_RC1	FMA1/2_(R)SAP_ACTIVATE.con (NO/IV) AD contains the M_status
LLI_FMA7_RC2	FMA1/2_SAP_DEACTIVATE.con (NO/IV) AD contains the M_status
LLI_FMA7_RC3	FDL_SEND_UPDATE.con (LS/LR/IV)/ FDL_REPLY_UPDATE.con (LS/LR/IV) AD contains the L_status
LLI_FMA7_RC4	FDL_CYC_ENTRY.con (LS/IV/NO) (activation) AD contains the L_status
LLI_FMA7_RC5	FDL_CYC_ENTRY.con (LS/IV/NO) (deactivation) AD contains the L_status
LLI_FMA7_RC6	Unallowed FDL primitive in the connection establishment phase or release phase AD contains the code of the primitive
LLI_FMA7_RC7	Unallowed FDL primitive in the data transfer phase, AD contains the code of the primitive
LLI_FMA7_RC8	unknown FDL primitive
LLI_FMA7_RC9	unknown LLI primitive
LLI_FMA7_RC10	Unallowed LLI primitive in the connection establishment phase or release phase AD contains the code of the primitive
LLI_FMA7_RC11	Unallowed LLI primitive in the data transfer phase, AD contains the code of the primitive
LLI_FMA7_RC12	FDL_DATA_ACK.con (LS/LR/IV) AD contains the L_status
LLI_FMA7_RC13	Error on transmission FDL_CYC_DATA_REPLY.con (LS/LR/IV/OK/NO) AD contains the L_status
LLI_FMA7_RC14	FDL_DATA_REPLY.con (LS/LR/IV) AD contains the L_status
LLI_FMA7_RC15	FDL_DATA.con (LS/LR/DS/IV) AD contains the L_status
LLI_FMA7_RC16	Error on receipt FDL_CYC_DATA_REPLY.con (LS/LR/IV/OK/NO) AD contains the L_status
LLI_FMA7_RC17	Error on transfer of the Poll List FDL_CYC_DATA_REPLY.con (LS/LR/IV/NO) AD contains the L_status
LLI_FMA7_RC18	Timer T1 expired (connection establishment)
LLI_FMA7_RC19	Timer T2 expired (connection release)
LLI_FMA7_RC20	Time out during Poll List activation/ deactivation
LLI_FMA7_RC21	LLI service primitive could not be assigned to the CRL
LLI_FMA7_RC22	Unallowed FMA1/2 primitive received AD contains the code of the primitive
LLI_FMA7_RC23	Unallowed FDL primitive received at start-up AD contains the code of the primitive
LLI_FMA7_RC24	Confirm / indication mode error

This page is intentionally left blank.

PROFIBUS Specification - Normative Parts
Part 7

Network Management

CONTENTS

1 Scope

see part 5

2 Normative References and additional Material

see part 5

3 General

see part 5

4 Fieldbus Management Layer 7 (FMA7)

4.1 Overview

As a supplement to the FMS services, FMA7 provides services for configuration of the bus and communication relationships, for supervision and diagnosis in the operational phase and for establishing and releasing a management connection.

The functions of FMA7 are arranged in three groups:

1. context management
2. configuration management
3. fault management

The functions of context management allow:

- establishment and release of a management connection

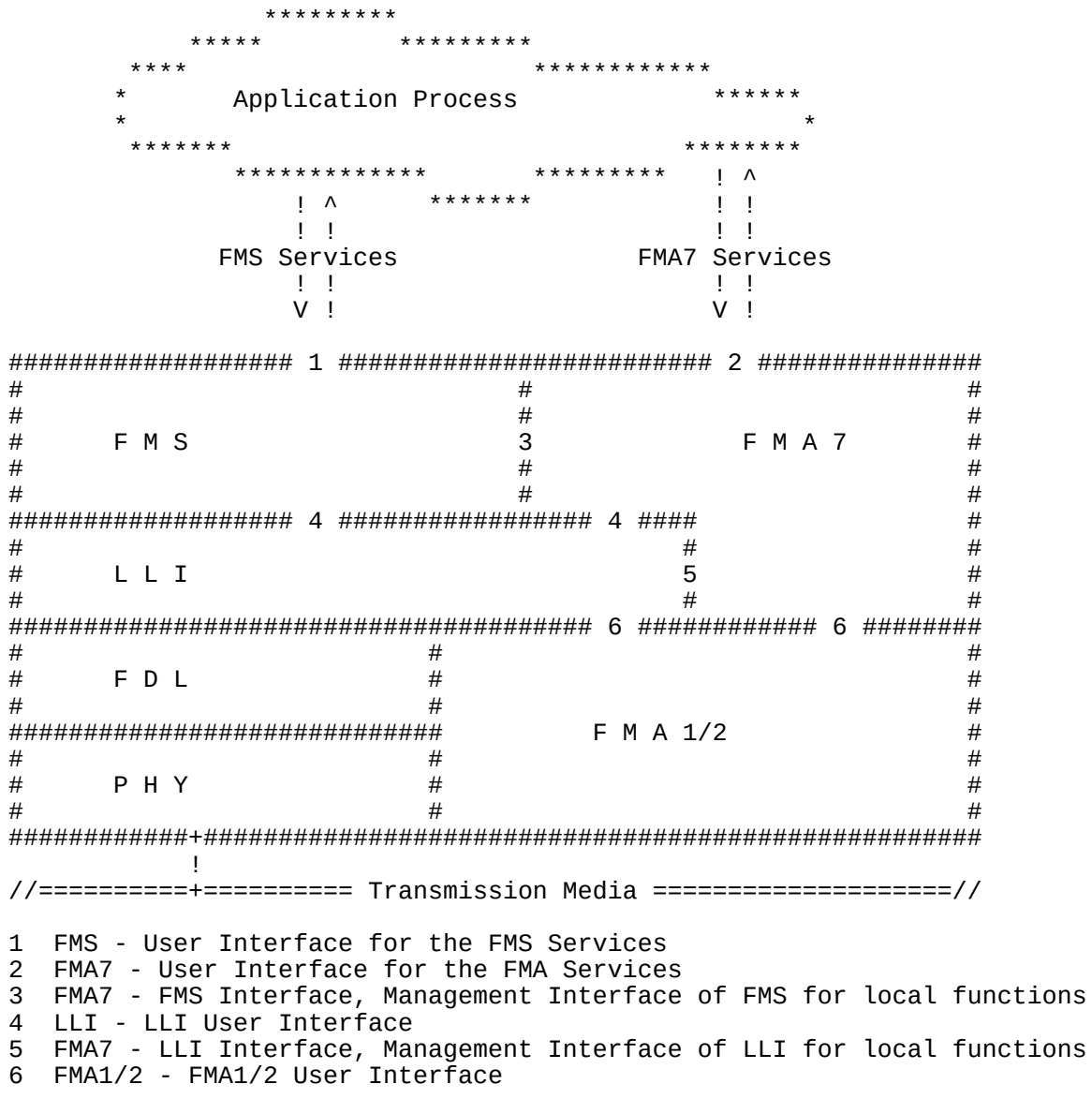
The functions of configuration management allow:

- loading and reading of the Communication Relationship List (CRL)
- access to variables, counters and parameters of Layers 1/2
- identification of communication components of stations
- registration of stations

The functions of fault management allow:

- indication of faults and events
- reset of stations

The following figure shows the placement of FMA7 in the ISO/OSI Layer Model and the interfaces to the other instances of the PROFIBUS Specification.



1 FMS - User Interface for the FMS Services
2 FMA7 - User Interface for the FMA Services
3 FMA7 - FMS Interface, Management Interface of FMS for local functions
4 LLI - LLI User Interface
5 FMA7 - LLI Interface, Management Interface of LLI for local functions
6 FMA1/2 - FMA1/2 User Interface

Figure 1. Placement of FMA7 in the ISO/OSI Layer Model

4.2 Features of FMA7

The FMA7 is based on system management according to ISO/DIS 7498-4:1989.

The functions of FMA7 are organized such that simple stations of a PROFIBUS System may be realized without a full management.

The functionality of FMA7 is adapted to the requirements of fieldbus and in particular supports a central configuration, which is independent of manufacturer, and supports the maintenance and starting-up of the fieldbus system.

FMA7 is based on the functions of FMA1/2 and on the management functions of LLI and FMS.

4.3 Model of FMA7

FMA7 describes the management objects, services and the resulting models. The objects are implicitly described (by the PROFIBUS Specification). The access to objects is realized with object specific services. It is distinguished between local and remote FMA7 services. FMA7 services may not be executed in parallel.

4.3.1 Local Management

The local management is characterized by the following features:

- The local management services allow the manipulation of local management objects in the individual instances.
- The local management services are mapped by FMA7 onto the management services of FMA1/2, LLI and FMS.

FMA7 provides the following local services:

InitiateLoad-CRL-Loc
Load-CRL-Loc
TerminateLoad-CRL-Loc
Read-CRL-Loc
SetValueLoc
ReadValueLoc
LSAP-StatusLoc
IdentLoc
GetLiveList
FMA7 Reset
FMA7 Event

The local management services are optional with the exception of the services FMA7 Event and FMA7 Reset.

4.3.2 Remote Management

The remote management is characterized by the following features:

- The remote management services allow the manipulation of remote management objects in the remote station.
- The remote management is connection-oriented and uses the transport functionality of LLI.
- The remote management is a user of LLI (as FMS) and uses the LLI SAP = 1 for the remote management services.
- The remote management services are characterized in that a service is performed by the remote user with the help of its local management.
- All PROFIBUS stations support remote management services as responder on exactly one management connection. The management connection is defined with regard to its properties and addressing at the responder of the remote management services (see subclause 5.2.2.1 Default Management Connection). Devices for configuration and diagnosis support remote management services also as a requester on several management connections.
- The FMA7 remote services are transmitted with FMA7 PDUs.

FMA7 provides the following remote services:

FMA7 Initiate
FMA7 Abort
InitiateLoad-CRL-Rem
Load-CRL-Rem
TerminateLoad-CRL-Rem
Read-CRL-Rem
SetValueRem

ReadValueRem
IdentRem
LSAP-StatusRem

All remote management services are optional.

Default Management Connection

The specification of the default management connection makes a standardized access to PROFIBUS stations possible for configuration or diagnosis devices. At every PROFIBUS station, which supports remote FMA7 services as a responder, an entry for a default management connection with CREF = 1 shall be entered in the CRL. The default management connection (responder side) occupies only one entry in the CRL. To allow a complete remote configuration of a station (Object Dictionary (OD), Communication Relationship List (CRL) and bus parameters), two connections for configuration shall be assigned in the CRL.

One management connection shall be for configuration of the CRL and the bus parameters and one FMS connection shall be for configuration of the OD.

The following CRL entry of a management connection is mandatory at the requester of remote management services (for structure and meaning of the CRL see CRL Object):

Attribute Name	Range of Values
CREF	2 .. FFFF hex
Local LSAP	0, 2 .. 62, NIL
Remote Address	0 .. 126
Remote LSAP	1
Type	MMAC/MSAC
LLI SAP	1
Connection Attribute	I/D

Figure 2. CRL Entry of a Management Connection at the Requester of Remote Management Services

The values of the other attributes in the CRL entry of the management connection are not defined by FMA7.

The following CRL entry of the management connection is mandatory at the responder of remote management services:

Attribute Name	Range of Values
CREF	1
Local LSAP	1
Remote Address	ALL
Remote LSAP	ALL
Type	MMAC/MSAC
LLI SAP	1
Connection Attribute	0

Figure 3. CRL Entry of the Management Connection at the Responder of Remote Management Services

The values of the other attributes in the CRL entry of the management connection are not defined by FMA7.

4.3.3 Formal Description of FMA7 State Machines

The formal description is based upon a model that describes the FMA7 protocol with the help of state machines (represented by state diagrams). The protocol sequences are described with different states (represented by rectangles with state names) and transitions between states (represented by lines with arrows). State changes are caused by events combined with conditions which cause actions (reactions).

The description of state transitions is similar to the draft DIN ISO 8802 Part 4. The elements used are the sequence (current state, event / condition => action, next state) as well as constants, variables, service primitives, functions and procedures.

The detailed description of state transitions and actions is structured for every state change as follows:

The first line defines the current state, the name of the transition and the next state. Below, in the subsequent lines, follow:

a) the events and conditions, which shall have become true for the transition to the following state and after that

b) the actions which are executed before entering the next state.

The events and conditions to be evaluated and the actions to be executed are described with a syntax based on the programming language PL/1.

4.4 Context Management

4.4.1 Model Description

The FMA7 context management services are used for establishing and releasing a management connection. The establishment of a management connection is always required when remote management services shall be executed. The context management services are mandatory for all devices which support remote management. If a PROFIBUS communication system does not support remote management the LSAP 1 shall not be activated.

4.4.2 The FMA7 CRL Object

The FMA7 Communication Relationship List (FMA7 CRL) contains the specific description of all management connections of a device independent of the time of use. The FMA7 CRL consists of the header, the static and the dynamic part. Information about the structure of the FMA7 CRL is entered into the FMA7 CRL header. The FMA7 CRL header is entered under CREF 0. The FMA7 CRL is structured in lines. Each line shall be addressed by the communication reference (CREF). Every line contains the complete FMA7-specific description of the respective management connection.

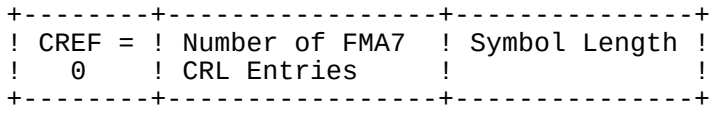


Figure 4. Structure of the FMA7 CRL Header

Communication Reference
The CRL header is addressed with CREF = 0.

Number of FMA7 CRL Entries
Specifies the number of occupied entries in the FMA7 CRL.

Symbol Length
This attribute specifies the length of symbols of the FMA7 CRL and may only have the values 0..32. The attribute shall be equal to the symbol length in the CRL header of FMS.

0 <=> no symbols

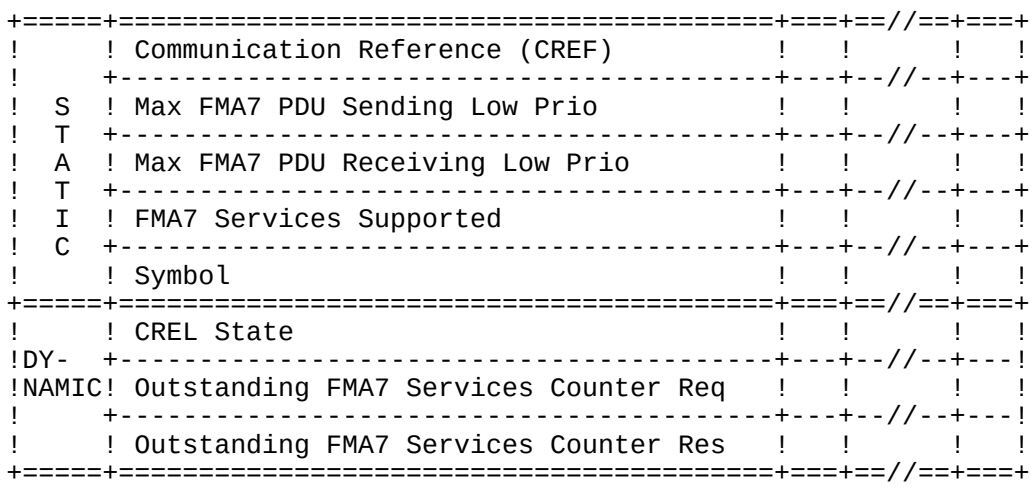


Figure 5. Structure of FMA7 CRL Entries

Communication Reference
The communication reference is a locally unique identifier for the communication relationship.

Max FMA7 PDU Sending Low Prio
This attribute contains the maximum possible length of the FMA7 PDU for sending with low priority which shall be handled on this communication relationship.

Max FMA7 PDU Receiving Low Prio
This attribute contains the maximum possible length of the FMA7 PDU for receiving with low priority which shall be handled on this communication relationship.

FMA7 Services Supported
This attribute gives information about which optional FMA7 services may be executed on this communication relationship. Two bits for every optional remote FMA7 service (service group) are used in a bit string to specify whether these services are supported as a request or as a response, respectively. If the corresponding bit is set, the services are supported as a request or as a response, respectively.

Table 1. Attribute FMA7 Services Supported

Service	Primitive bit[n]	Primitive bit[m]
reserved *)	.req,.con 0	.ind,.res 8
InitiateLoad-CRL-Rem	.req,.con 1	.ind,.res 9
Load-CRL-Rem,	.req,.con "	.ind,.res "
TerminateLoad-CRL-Rem	.req,.con "	.ind,.res "
Read-CRL-Rem	.req,.con 2	.ind,.res 10
SetValueRem	.req,.con 3	.ind,.res 11
ReadValueRem	.req,.con 4	.ind,.res 12
LSAP-StatusRem	.req,.con 5	.ind,.res 13
IdentRem	.req,.con 6	.ind,.res 14
reserved *)	.req,.con 7	.ind,.res 15
reserved *)	16 to 31	32 to 47

Explanation:
*) reserved bits shall be set to 0
n = 0 to 7, 16 to 31
m = 8 to 15, 32 to 47

Symbol
The symbolic name of the communication reference. The existence and the length are specified in the CRL header.

CREL State
This attribute contains the state of the communication relation-ship.

The following states are permissible:

- CONNECTION-NOT-ESTABLISHED
- CONNECTION-ESTABLISHING (CALLING)
- CONNECTION-ESTABLISHING (CALLED)
- CONNECTION-ESTABLISHED

Outstanding FMA7 Services Counter Req

This attribute specifies how many confirmed services (remote FMA7 services) are currently pending at the requester on this communication relationship.

Outstanding FMA7 Services Counter Res

This attribute specifies how many outstanding confirmed services (remote FMA7 services) are being handled at the responder on this communication relationship.

Attributes

Object: FMA7 Communication Relationship List
  Key Attribute: implicit
  Attribute: FMA7 CRL Header
  Attribute: List of FMA7 CRL Entry
  Attribute: CRL Loader State
  Attribute: CRL Upload State

Object: FMA7 CRL Header
  Key Attribute: Communication Reference
  Attribute: Number of FMA7 CRL Entries
  Attribute: Symbol Length

Object: FMA7 CRL Entry
  Key Attribute: Communication Reference
  Attribute: Max FMA7 PDU Sending Low Prio
  Attribute: Max FMA7 PDU Receiving Low Prio
  Attribute: FMA7 Services Supported
  Attribute: Symbol
  Attribute: CREL State
  Attribute: Outstanding FMA7 Services Counter Req
  Attribute: Outstanding FMA7 Services Counter Res

4.4.3 Services

4.4.3.1 FMA7 Initiate

The FMA7 user shall use the FMA7 Initiate service to establish a management connection.

Table 2. FMA7 Initiate Service

Parameter Name	.req	.ind	.res	.con
Argument	M	M=		
Communication Reference	M	M		
Max FMA7 PDU Sending Low Prio *)				
Max FMA7 PDU Receiving Low Prio *)				
FMA7 Services Supported *)				
Result(+)			S	S=
Communication Reference			M	M
Result(-)			S	S=
Communication Reference			M	M=
Error Code			M	M
Max FMA7 PDU Sending Low Prio **)				M
Max FMA7 PDU Receiving Low Prio **)				M
FMA7 Services Supported **)				M

Explanation:
*) Calling
**) Called

Argument
The argument contains the parameters of the FMA7 Initiate.req primitive and the FMA7 Initiate.ind primitive.

Communication Reference
This parameter specifies the identifier of the associated management connection in the CRL.

Max FMA7 PDU Sending Low Prio (Calling)
This parameter contains the maximum permissible length of the FMA7 PDU for sending with low priority which shall be handled on this communication relationship. It is transmitted by the calling FMA7 and is not a part of the interface primitives.

Max FMA7 PDU Receiving Low Prio (Calling)
This parameter contains the maximum permissible length of the FMA7 PDU for receiving with low priority which shall be handled on this communication relationship. It is transmitted by the calling FMA7 and is not a part of the interface primitives.

FMA7 Services Supported (Calling)
This parameter specifies which FMA7 services may be executed by the client (see FMA7 CRL). It is transmitted by the calling FMA7 and is not a part of the interface primitives.

Result(+)
The Result(+) parameter indicates that the Initiate service was executed suc-

cessfully.

Result(-)
The Result(-) parameter indicates that the Initiate service failed.

Error Code
This parameter contains information about the reason the service failed.

- Other
  The error code could not be assigned to any of the error codes specified below.
- Max PDU Size Insufficient
  The specified maximum PDU length is not sufficient for the requested communication.
- Service Not Supported
  The requested service is not supported by the server.
- User Initiate Denied
  The FMA7 user rejects the FMA7 Initiate.

Max FMA7 PDU Sending Low Prio (Called)
This parameter contains the maximum permissible length of the FMA7 PDU for sending with low priority which shall be handled on this communication relationship. It is transmitted by the called FMA7 and is a part only of the interface primitive initiate.con.

Max FMA7 PDU Receiving Low Prio (Called)
This parameter contains the maximum permissible length of the FMA7 PDU for receiving with low priority which shall be handled on this communication relationship. It is transmitted by the called FMA7 and is a part only of the interface primitive initiate.con.

FMA7 Services Supported (Called)
This parameter specifies which FMA7 services may be executed by the server (see FMA7 CRL). It is not a part of the interface primitive initiate.res.

4.4.3.2 FMA7 Abort

The FMA7 user shall use this service to release an existing connection.

Table 3. FMA7 Abort Service

Parameter Name	.req	.ind
Argument	M	M
Communication Reference	M	M
Locally Generated		M
Abort Identifier	M	M=
Reason Code	M	M=
Abort Detail	U	C

Argument
The argument contains the parameters of the FMA7 Abort.req primitive and the FMA7 Abort.ind primitive.

Communication Reference
This parameter specifies the identifier of the associated management connection in the CRL.

Locally Generated

This parameter indicates whether the abort was initiated locally or by the communication partner.

TRUE - The abort was initiated locally.

FALSE - The abort was initiated remotely.

Abort Identifier

This parameter indicates where the reason for the connection abort was identified.

Valid values

- User
- LLI User (FMA7)
- LLI
- Layer 2

Reason Code

This parameter specifies the reason for the connection abort.

If the parameter Abort Identifier has the value LLI or Layer 2, then the parameter Reason Code is generated in LLI and shall be found in the description of LLI.

If the parameter Abort Identifier has the value User, the following value is specified:

- ABT_RC1 <=> DISCONNECT
  User releases the connection

If the parameter Abort Identifier has the value FMA7, the following values are specified:

- ABT_RC1 <=> FMA7-CRL-Error
  FMA7 CRL Entry incorrect

- ABT_RC2 <=> User Error
  Invalid, unknown or faulty service primitive received by the FMA7 user

- ABT_RC3 <=> FMA7-PDU-Error
  Invalid or unknown FMA7 PDU received by LLI

- ABT_RC4 <=> Connection-State-Conflict-LLI
  Invalid LLI service primitive

- ABT_RC5 <=> LLI-Error
  Faulty or unknown LLI service primitive

- ABT_RC6 <=> PDU-Size
  PDU length > maximum PDU length

- ABT_RC7 <=> FMA7-Service-Not-Supported
  FMA7 service.req received by the FMA7 user and the service is not supported as client
  or
  FMA7_SERVICE_REQ_PDU received by LLI and the service is not supported as server
  (see attribute FMA7 Services Supported of the FMA7 CRL)

- ABT_RC8 <=> Response-Error
  FMA7 service.res received by the FMA7 user and Outstanding FMA7 Service Counter Res = 0
  or

FMA7-SERVICE_RES_PDU received by LLI and Outstanding FMA7 Service Counter Req = 0

- ABT_RC9 <=> Max-Services-Overflow
  Number of allowed parallel services exceeded

- ABT_RC10 <=> Connection-State-Conflict-FMA7
  FMA7-INITIATE_REQ_PDU received by LLI

- ABT_RC11 <=> Service-Error
  The service in the response does not correspond to the service in the indication.
  or
  The service in the confirmation does not correspond to the service in the request.

- ABT_RC12 <=> Loading CRL
  The CRL is currently loaded.

If the parameter Abort Identifier has the value LLI or Layer 2, the Reason Code is generated in LLI.

The coding of the Abort_PDU is specified in LLI.

4.4.4 Context Test in FMA7

Upon receiving a FMA7_INITIATE_REQ_PDU the FMA7 of the responder tests the FMA7 context of the communication partner (remote context) for compatibility to its own FMA7 context (local context) for this connection entered in the CRL. The local FMA7 context is assumed to be correctly configured. The compatibility of the FMA7 contexts may be deduced from the following matrix (see Fig. 196). For parameters of different meaning (e.g. the FMA7 Services Supported parameter in the local context and the FMA7 Outstanding Services Parameter in the remote context) a test is not required. The corresponding entries in the matrix remain empty.

	Local Context FMA7			
	Max FMA7 PDU		FMA7	
Remote Context FMA7	Sending Low Prio	Receiving Low Prio	Services Supported	
			.req	.res
			[n]	[m]
	.req,.res	.ind,.con	0 1	0 1
Max FMA7 PDU Sending Low Prio	.req,.res	•		
Max FMA7 PDU Receiving Low Prio	.ind,.con	•		
FMA7 Services Supported	.req [n] 0			X X
	.req [n] 1			- X
	.res [m] 0		X -	
	.res [m] 1		X X	

Explanation:
X : compatible
\- : not compatible (Error case)
• : local value smaller than or equal to remote value
• : local value larger than or equal to remote value
.req 0 : Service was not supported as requester
.req 1 : Service was supported as requester
.res 0 : Service was not supported as responder
.res 1 : Service was supported as responder
[n] : 0 to 7, 16 to 23
[m] : 8 to 15, 24 to 47

Figure 6. Compatibility of the local FMA7 Context to the remote FMA7 Context

4.4.5 State Machine of a Management Connection

4.4.5.1 State Machine Description

CONNECTION-NOT-ESTABLISHED

No FMA7 context is established. Only the service primitives FMA7 Initiate.req, ASS.ind, FMA7 Abort.req and ABT.ind are permitted. All other services are rejected.

CONNECTION-ESTABLISHING (CALLING)

The local FMA7 user wishes to establish a connection. Only the service primitives ASS.con(+), ASS.con(-), FMA7 Abort.req and ABT.ind are permitted. All other services are rejected. In the following this state is abbreviated with CON-ESTABLISHING-CALLING.

CONNECTION-ESTABLISHING (CALLED)

The remote FMA7 user wishes to establish a connection. Only the service primitives FMA7 Initiate.res(+), FMA7 Initiate.res(-), ABT.ind and FMA7 Abort.req are permitted. All other services are rejected. In the following this state is abbreviated with CON-ESTABLISHING-CALLED.

CONNECTION-ESTABLISHED

The communication relationship is established. The service primitives FMA7 Initiate.req, FMA7 Initiate.res(+), FMA7 Initiate.res(-) are not allowed and are rejected. No parallel remote FMA7 services are allowed. Only the LLI service primitives DTC.ind, DTC.con and ABT.ind are allowed. All other are rejected. ASS.ind shall be rejected with the separate error message "Connection State Conflict FMA7".

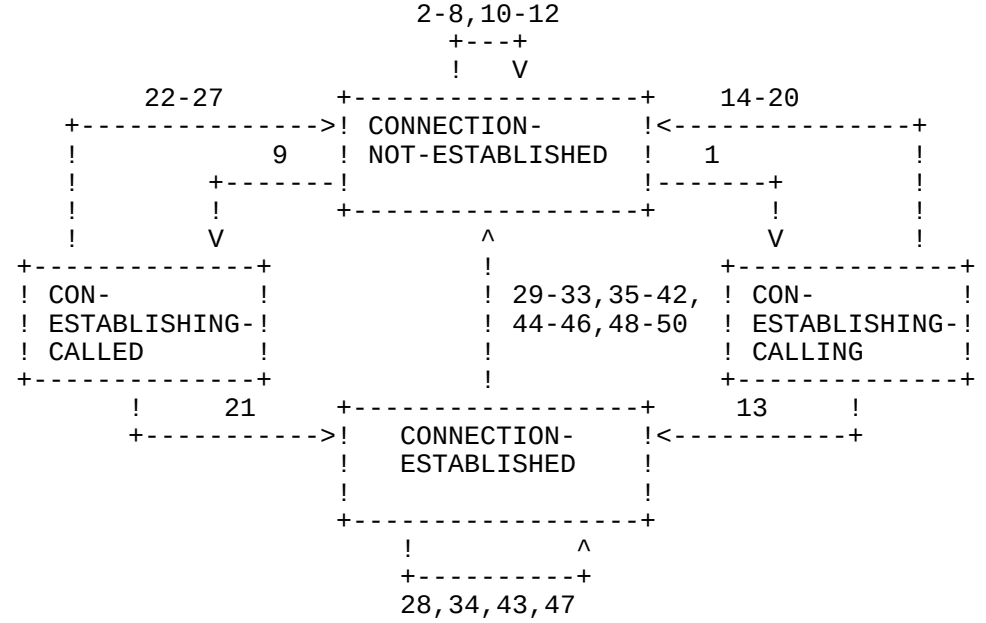


Figure 7. State Machine

For the reset of a communication reference (CREF reset) the following actions shall be executed:

- clearing of memory content
- in the dynamic part of the FMA7 CRL the fields 'Outstanding FMA7 Service Counter Req' and 'Outstanding FMA7 Service Counter Res' are set to 0.
- the CREL State is set to 'CONNECTION-NOT-ESTABLISHED'

4.4.5.2 State Transitions

Connection Establishment and Release

Current State / Event / \Exit Condition / => Action Taken	Transition	Next State
CONNECTION-NOT-ESTABLISHED FMA7 Initiate.req received from FMA7 user \FMA7 CRL entry valid => send FMA_INITIATE_REQ_PDU, (ASS.req) to LLI	**1**	**CON-ESTABLISHING-CALLING**
CONNECTION-NOT-ESTABLISHED FMA7 Initiate.req received from FMA7 user \FMA7 CRL entry invalid => FMA7 Abort.ind to FMA7 user <reason code = ABT_RC1>	**2**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED FMA7 Abort.req received from FMA7 user => ignore	**3**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED unknown, unallowed or faulty service primitive received from FMA7 user => FMA7 Abort.ind to FMA7 user <reason code = ABT_RC2>	**4**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED ABT.ind from LLI received => ignore	**5**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED unknown or faulty service primitive received from LLI => ABT.req to LLI <reason code = ABT_RC5>	**6**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED unknown or unallowed LLI service primitive received from LLI => ABT.req to LLI <reason code = ABT_RC4>	**7**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED not allowed or faulty FMA7 PDU (ASS.ind) received => ABT.req to LLI <reason code = ABT_RC3>	**8**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED FMA7_INITIATE_REQ_PDU(ASS.ind) received from LLI \FMA7 CRL entry is valid AND FMA7 context test positive => FMA7 Initiate.ind to FMA7 user	**9**	**CON-ESTABLISHING-CALLED**
CONNECTION-NOT-ESTABLISHED FMA7_INITIATE_REQ_PDU(ASS.ind) from LLI received \FMA7 CRL entry is invalid => ABT.req to LLI <reason code = ABT_RC1>	**10**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-NOT-ESTABLISHED FMA7_INITIATE_REQ_PDU (ASS.ind) received from LLI \FMA7 CRL entry is valid AND max FMA7 PDU length test negative => send FMA7_INITIATE_ERROR_PDU, (ASS.res(-)) to LLI <error code = 1>	**11**	**CONNECTION-NOT-ESTABLISHED**

Connection Establishment and Release

Current State / Event / \Exit Condition / => Action Taken	Transition	Next State
CONNECTION-NOT-ESTABLISHED FMA7_INITIATE_REQ_PDU (ASS.ind) received from LLI \FMA7 CRL entry is valid AND max FMA7 PDU length test positive AND FMA7 services supported test negative => send FMA7_INITIATE_ERROR_PDU, (ASS.res(-)) to LLI <error code = 2>	**12**	**CONNECTION-NOT-ESTABLISHED**
CON-ESTABLISHING-CALLING FMA7_INITIATE_RES_PDU (ASS.con(+)) received from LLI => FMA7 Initiate.con(+) to FMA7 user	**13**	**CONNECTION-ESTABLISHED**
CON-ESTABLISHING-CALLING INITIATE-ERROR_PDU (ASS.con(-) from LLI) => FMA7 Initiate.con(-) to FMA7 user, reset CREF	**14**	**CONNECTION-NOT-ESTABLISHED**
CON-ESTABLISHING-CALLING ABT.ind received from LLI => FMA7 Abort.ind to FMA7 user <reason code out of ABT.ind>, reset CREF	**15**	**CONNECTION-NOT-ESTABLISHED**
CON-ESTABLISHING-CALLING FMA7 Abort.req received from FMA7 user => ABT.req to LLI <reason code as given by user> reset CREF	**16**	**CONNECTION-NOT-ESTABLISHED**
CON-ESTABLISHING-CALLING unknown or faulty service primitive received from LLI => ABT.req to LLI <reason code = ABT_RC5> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC5> reset CREF	**17**	**CONNECTION-NOT-ESTABLISHED**
CON-ESTABLISHING-CALLING unallowed LLI service primitive received from LLI => ABT.req to LLI <reason code = ABT_RC4> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC4> reset CREF	**18**	**CONNECTION-NOT-ESTABLISHED**
CON-ESTABLISHING-CALLING unknown, unallowed or faulty FMA7 PDU (ASS.con) received from LLI => ABT.req to LLI <reason code = ABT_RC3> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC3> reset CREF	**19**	**CONNECTION-NOT-ESTABLISHED**
CON-ESTABLISHING-CALLING unknown, unallowed or faulty service primitive received from FMA7 user => ABT.req to LLI <reason code = ABT_RC2> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC2> reset CREF	**20**	**CONNECTION-NOT-ESTABLISHED**

Connection Establishment and Release

Current State / Event / \Exit Condition / => Action Taken	Transition	Next State
CON-ESTABLISHING-CALLED FMA7 Initiate.res(+) received from FMA7 user => send FMA7_INITIATE_RES_PDU, (ASS.res(+)) to LLI	**21**	**CONNECTION-ESTABLISHED**
CON-ESTABLISHING-CALLED FMA7 Initiate.res(-) received from FMA7 user => send FMA7_INITIATE_ERROR_PDU, (ASS.res(-)) to LLI reset CREF	**22**	**CONNECTION-NOT-ESTABLISHED**
CON-ESTABLISHING-CALLED FMA7 Abort.req received from FMA7 user => ABT.req to LLI <reason code as given by user> reset CREF	**23**	**CONNECTION-NOT-ESTABLISHED**
CON-ESTABLISHING-CALLED ABT.ind received from LLI => FMA7 Abort.ind to FMA7 user <reason code out of ABT.ind>, reset CREF	**24**	**CONNECTION-NOT-ESTABLISHED**
CON-ESTABLISHING-CALLED unknown or faulty service primitive received from LLI => ABT.req to LLI <reason code = ABT_RC5> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC5> reset CREF	**25**	**CONNECTION-NOT-ESTABLISHED**
CON-ESTABLISHING-CALLED unallowed LLI service primitive received from LLI => ABT.req to LLI <reason code = ABT_RC4> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC4> reset CREF	**26**	**CONNECTION-NOT-ESTABLISHED**
CON-ESTABLISHING-CALLED unknown, unallowed or faulty service primitive received from FMA7 user => ABT.req to LLI <reason code = ABT_RC2> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC2> reset CREF	**27**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED FMA7 service Rem.req received from FMA7 user \outstanding FMA7 service counter req= 0 AND PDU length • max FMA7 PDU sending low prio AND FMA7 services supported test(Client) positive => FMA7_SERVICE_REQ_PDU with DTC.req to LLI outstanding FMA7 service counter req := 1	**28**	**CONNECTION-ESTABLISHED**
CONNECTION-ESTABLISHED FMA7 service Rem.req received from FMA7 user \outstanding FMA7 service counter req • 1 => ABT.req to LLI <reason code = ABT_RC9> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC9> reset CREF	**29**	**CONNECTION-NOT_ESTABLISHED**

Connection Establishment and Release

Current State Event \Exit Condition => Action Taken	Transition	Next State
CONNECTION-ESTABLISHED FMA7 service Rem.req received from FMA7 user \outstanding FMA7 service counter req = 0 AND PDU length > max FMA7 PDU sending low prio => ABT.req to LLI <reason code = ABT_RC6> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC6> reset CREF	**30**	**CONNECTION-NOT_ESTABLISHED**
CONNECTION-ESTABLISHED FMA7 service Rem.req received from FMA7 user received \outstanding FMA7 service counter req = 0 AND PDU length ≤ max FMA7 PDU sending low prio AND FMA7 services supported test (Client) negative => ABT.req to LLI <reason code = ABT_RC7> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC7> reset CREF	**31**	**CONNECTION-NOT_ESTABLISHED**
CONNECTION-ESTABLISHED FMA7 Abort.req received from FMA7 user received => ABT.req to LLI <reason code as given by user> reset CREF	**32**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED unknown, faulty or unallowed service primitive received from FMA7 user => ABT.req to LLI <reason code = ABT_RC2> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC2> reset CREF	**33**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED FMA7_SERVICE_REQ_PDU (DTC.ind) received from LLI \PDU length • max FMA7 PDU receiving low prio AND outstanding FMA7 service counter res = 0 AND FMA7 services supported test (Server) positive => FMA7 service Rem.ind to FMA7 user outstanding FMA7 service counter res := 1	**34**	**CONNECTION-ESTABLISHED**
CONNECTION-ESTABLISHED FMA7_SERVICE_REQ_PDU (DTC.ind) received from LLI \PDU length > max FMA7 PDU receiving low prio => ABT.req to LLI <reason code = ABT_RC6> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC6> reset CREF	**35**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED FMA7_SERVICE_REQ_PDU (DTC.ind) received from LLI \PDU length • max FMA7 PDU receiving low prio AND outstanding FMA7 service counter res ≥ 1 => ABT.req to LLI <reason code = ABT_RC9> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC9> reset CREF	**36**	**CONNECTION-NOT-ESTABLISHED**

Connection Establishment and Release

Current State / Event / \Exit Condition / => Action Taken	Transition	Next State
CONNECTION-ESTABLISHED FMA7_SERVICE_REQ_PDU (DTC.ind) received from LLI \PDU length • max FMA7 PDU receiving low prio AND outstanding FMA7 service counter res = 0 AND FMA7 services supported test (Server) negative => ABT.req to LLI <reason code = ABT_RC7> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC7> reset CREF	**37**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED ABT.ind received from LLI => FMA7 Abort.ind to FMA7 user <reason code out of ABT.ind> reset CREF	**38**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED FMA7_INITIATE_REQ_PDU (ASS.ind) received from LLI => ABT.req to LLI <reason code = ABT_RC10> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC10> reset CREF	**39**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED unknown or faulty service primitive received from LLI => ABT.req to LLI <reason code = ABT_RC5> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC5> reset CREF	**40**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED unallowed LLI service primitive received from LLI => ABT.req to LLI <reason code = ABT_RC4> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC4> reset CREF	**41**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED unknown, unallowed or faulty FMA7 PDU (DTC.ind/ DTC.con/ASS.ind) received from LLI received => ABT.req to LLI <reason code = ABT_RC3> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC3> reset CREF	**42**	**CONNECTION-NOT-ESTABLISHED**
CONNECTION-ESTABLISHED FMA7 service Rem.res received from FMA7 user \outstanding FMA7 service counter res =1 AND service of .res is identical with service of .ind AND PDU length • max FMA7 PDU sending low prio => FMA7_SERVICE_RES_PDU with DTC.res to LLI outstanding FMA7 service counter res := 0	**43**	**CONNECTION-ESTABLISHED**
CONNECTION-ESTABLISHED FMA7 service Rem.res received from FMA7 user \outstanding FMA7 service counter res = 0 => ABT.req to LLI <reason code = ABT_RC8> FMA7 Abort.ind to FMA7 user <reason code = ABT_RC8> reset CREF	**44**	**CONNECTION-NOT-ESTABLISHED**

Connection Establishment and Release

Current State	Transition	Next State
Event \Exit Condition => Action Taken		

CONNECTION-ESTABLISHED **45** **CONNECTION-NOT-ESTABLISHED**

FMA7 service Rem.res received from FMA7 user
\outstanding FMA7 service counter res = 1
AND service of .res is not identical with
service of .ind
=> ABT.req to LLI <reason code = ABT_RC11>
FMA7 Abort.ind to FMA7 user <reason code = ABT_RC11>
reset CREF

CONNECTION-ESTABLISHED **46** **CONNECTION-NOT-ESTABLISHED**

FMA7 service Rem.res received from FMA7 user
\outstanding FMA7 service counter res = 1
AND service of .res is identical with
service of .ind
AND PDU length > max FMA7 PDU sending low prio
=> ABT.req to LLI <reason code = ABT_RC6>
FMA7 Abort.ind to FMA7 user <reason code = ABT_RC6>
reset CREF

CONNECTION-ESTABLISHED **47** **CONNECTION-ESTABLISHED**

FMA7_SERVICE_RES_PDU (DTC.con) received from LLI
\PDU length ≤ max FMA7 PDU receiving low prio
AND outstanding FMA7 service counter req = 1
AND service of .con is identical with
service of .req
=> FMA7 service Rem.con to FMA7 user
outstanding FMA7 service counter req := 0

CONNECTION-ESTABLISHED **48** **CONNECTION-NOT-ESTABLISHED**

FMA7_SERVICE_RES_PDU (DTC.con) received from LLI
\PDU length> max FMA7 PDU receiving low prio
=> ABT.req to LLI <reason code = ABT_RC6>
FMA7 Abort.ind to FMA7 user <reason code = ABT_RC6>
reset CREF

CONNECTION-ESTABLISHED **49** **CONNECTION-NOT-ESTABLISHED**

FMA7_SERVICE_RES_PDU (DTC.con) received from LLI
\PDU length • max FMA7 PDU receiving low prio
AND outstanding FMA7 service counter req = 0
=> ABT.req to LLI <reason code = ABT_RC8>
FMA7 Abort.ind to FMA7 user <reason code = ABT_RC8>
reset CREF

CONNECTION-ESTABLISHED **50** **CONNECTION-NOT-ESTABLISHED**

FMA7_SERVICE_RES_PDU (DTC.con) received from LLI
\PDU length • max FMA7 PDU receiving low prio
AND outstanding FMA7 service counter req >= 1
AND service of .con is identical with
service of .req
=> ABT.req to LLI <reason code = ABT_RC11>
FMA7 Abort.ind to FMA7 user <reason code = ABT_RC11>
reset CREF

4.5 Configuration Management

4.5.1 CRL Management

4.5.1.1 Model Description

The CRL management services are required to read and write the CRL of the PROFIBUS stations. For writing, FMA7 divides a complete CRL entry into the FMS or the FMA7 CRL entry and the LLI CRL entry and loads the parts into the corresponding instances. For reading, FMA7 combines the FMS or the FMA7 CRL entry and the LLI CRL entry to a complete CRL entry and delivers this to the FMA7 user.

4.5.1.2 The CRL Object

The CRL contains the description of all communication relation-ships of a device independent of the time of use. The CRL contains one CRL header and one CRL entry for each configured communication relationship. The CRL header and every CRL entry are addressed by the communication reference (CREF). The entries of the CRL shall be entered in ascending order without gaps. Every CRL entry contains the complete description of the corresponding communication relationship.

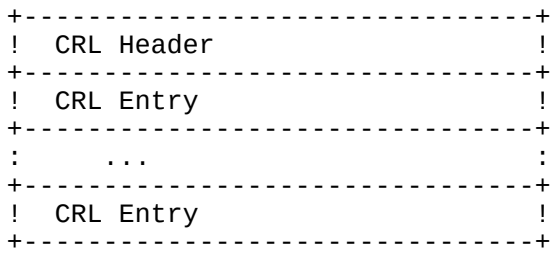


Figure 8. Structure of the CRL

The CRL Header Object

Information about the structure of the CRL is entered in the CRL header. The CRL header is entered under the communication reference 0.

CREF (=0)	Number of CRL Entries	Poll LIST SSAP	ASS/ABT CI	Symbol Length	VFD Pointer Supported

Figure 9. Structure of the CRL Header

Communication Reference
The CRL header is addressed with the communication reference = 0.

Number of CRL Entries
Specifies the number of entries in the CRL additional to the CRL header.

Poll List SSAP
Specifies the local Service Access Point which contains the Poll List of Layer 2.

Valid values
(0, 2..62) - SAP values
(128) - NIL
(255) - No Poll List SSAP used in current CRL

ASS/ABT CI

Contains the length of the time interval for the control of the connection establishment and the connection release. The value of ASS/ABT CI shall be multiplied by 10 ms to determine the control interval for the idle control.

Range of values: 0 ms to 10 · (2**32 - 1) ms.

Symbol Length

This attribute specifies the length of symbols in the CRL and may only take the values 0 to 32. If this attribute has the value 0, then the attribute "Symbol" does not exist in the CRL entry.

VFD Pointer Supported

This attribute specifies whether several VFDs are supported in the CRL.

FALSE - Only one VFD is supported.

TRUE - Several VFDs are supported.

The CRL Entry Object

Every CRL entry is addressed by the communication reference. Every CRL entry consists of a static and a dynamic part. The static part consists of 20 attributes and contains the complete description of the associated communication relationship. Only this part is loaded and therefore configured. The optional User Extension can only be loaded and read remotely and contains CREF related configuration information for the FMA7-user.

The dynamic part of a CRL entry consists of 8 attributes and can be optionally transferred in addition to the static part when the CRL is read.

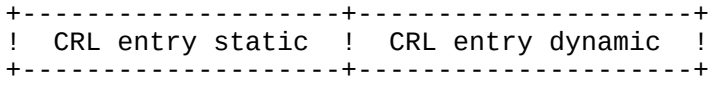


Figure 10. Structure of a CRL Entry

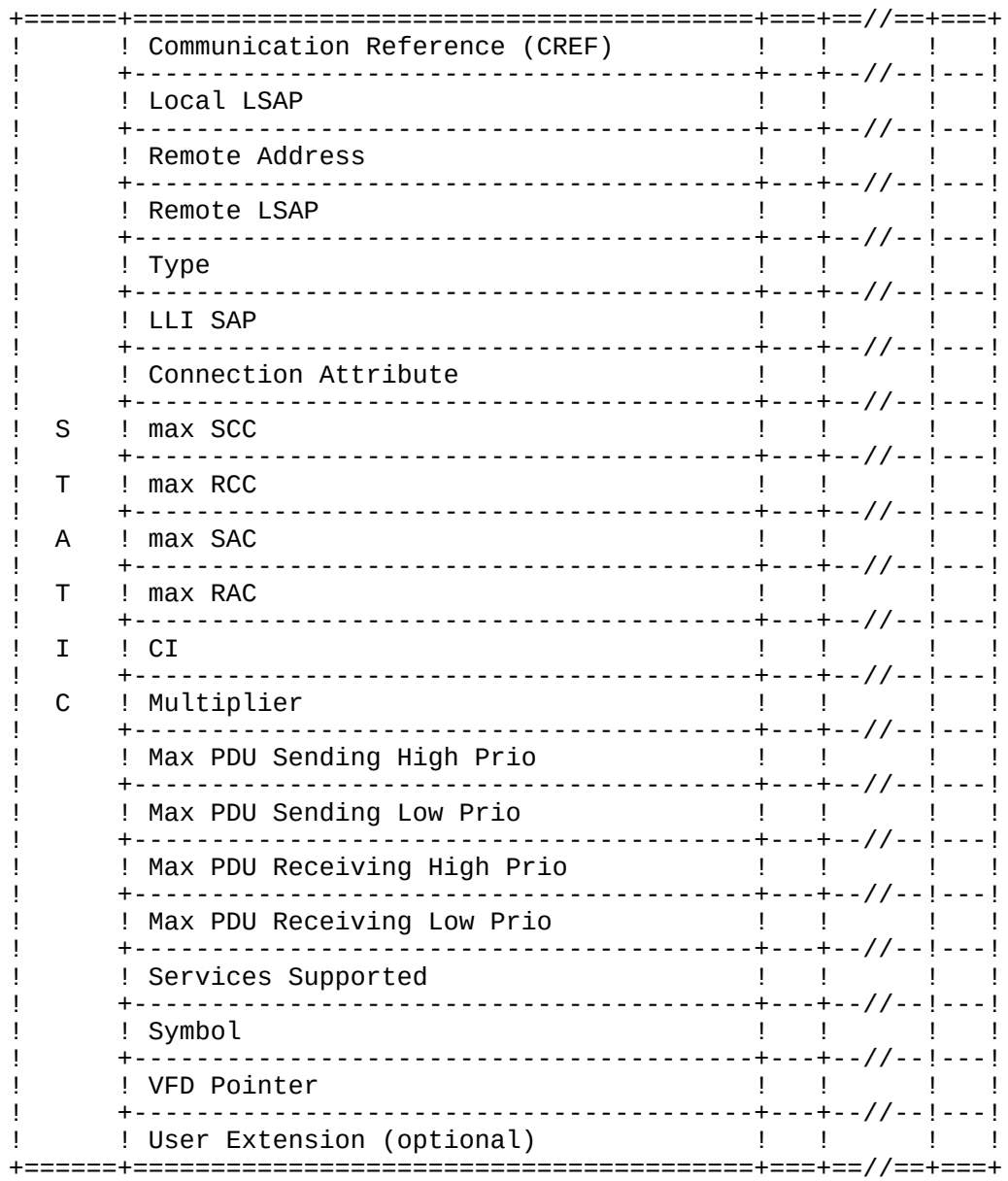


Figure 11. Structure of a static CRL Entry

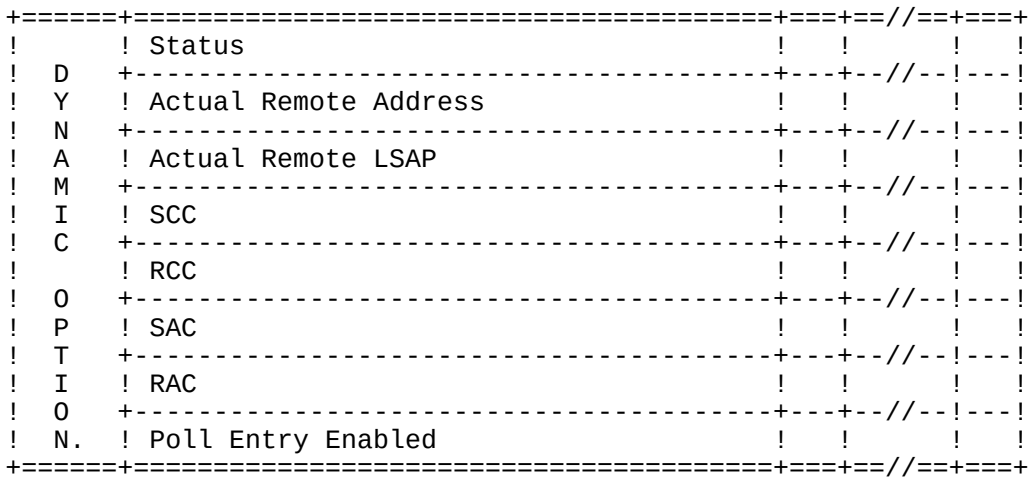


Figure 12. Structure of a dynamic CRL Entry

The range of values is specified by FMS, LLI or Layers 1/2, if not given here.

Description of the attributes for the static part of a CRL entry:

Communication Reference
The communication reference is a locally unique identifier of the communication relationship.

Local LSAP
Specifies the local Service Access Point which is used for this communication relationship.

Valid values
(0..63) - SAP values
(128) - NIL

Remote Address
Specifies the FDL address of the remote station for this communication relationship.

Valid values
(0..127) - station address
(255) - ALL

Remote LSAP
Specifies the Service Access Point of the remote station which is used for this communication relationship.

Valid values
(0..63) - SAP values
(128) - NIL
(255) - ALL

Type
Specifies the type of this communication relationship.

Valid values
(0) - MMAC master-master connection for acyclic data transfer
(1) - MSAC master-slave connection for acyclic data transfer with no slave initiative

(5) - MSAC_SI master-slave connection for acyclic data transfer with slave initiative
(3) - MSCY master-slave connection for cyclic data transfer with no slave initiative
(7) - MSCY_SI master-slave connection for cyclic data transfer with slave initiative
(8) - BRCT broadcast communication relationship

(10) - MULT multicast communication relationship

LLI SAP
This attribute specifies the configured LLI user for this communication relationship.

Valid values
(0) - FMS
(1) - FMA7

Connection Attribute
Contains further information on the connection type for connection-oriented communication relationships.

Valid values
(0) - D defined connection (master-master or master-slave connection)
(1) - I open connection at the requester (master-master connection)
(2) - O open connection at the responder (master-master or master-slave connection)

max SCC (maximum value Send Confirmed Request Counter)

This attribute specifies the maximum permitted number of parallel confirmed services to the communication partner. The attribute Outstanding Services Client in the FMS CRL shall be set equal to this attribute in case of a connection for acyclic data transfer. In case of a connection for cyclic data transfer the FMA7 shall assign the value 1 to the attribute Outstanding Services Client in the FMS CRL.

max RCC (maximum value Receive Confirmed Request Counter)
This attribute specifies the maximum permitted number of parallel confirmed services from the communication partner. The attribute Outstanding Services Server in the FMS CRL shall be set equal to this attribute in case of a connection for acyclic data transfer. In case of a connection for cyclic data transfer the FMA7 shall assign the value 1 to the attribute Outstanding Services Server in the FMS CRL.

max SAC (maximum value Send Acknowledged Request Counter)
This attribute specifies the maximum permitted number of parallel unconfirmed services to the communication partner.

max RAC (maximum value Receive Acknowledged Request Counter)

This attribute specifies the maximum permitted number of parallel unconfirmed services from the communication partner.

CI (Control Interval)
This attribute specifies the length of the time interval for the connection control on connection-oriented communication relationships. The value of CI shall be multiplied by 10 ms to determine the control interval. For connections for acyclic data transfer it includes the interval of the idle control (ACI) and for connections for cyclic data transfer the interval of the connection control (CCI). Range of values: 0 ms to 10 · (2**32 - 1) ms.

Multiplier
This attribute specifies for connections for cyclic data transfer on the master side how often the Layer 2 address of this communication relationship shall be entered into the poll list. Thereby the poll interval may be shortened.

Max PDU Sending High Prio
This attribute contains the maximum permissible length of a FMS PDU for sending with high priority which may be handled on this communication relationship.

Max PDU Sending Low Prio
This attribute contains the maximum permissible length of a FMS PDU for sending with low priority which may be handled on this communication relationship.

Max PDU Receiving High Prio
This attribute contains the maximum permissible length of a FMS PDU for receiving with high priority which may be handled on this communication relationship.

Max PDU Receiving Low Prio

This attribute contains the maximum permissible length of a FMS PDU for receiving with low priority which may be handled on this communication relationship.

Services Supported
This attribute contains the content of FMA7 Services Supported in the case of a management connection. In the case of a FMS connection, this attribute contains the content of FMS Features Supported.

- FMS Features Supported
This attribute specifies which FMS services and options are supported on this communication relationship. Two bits for each of the services are used to specify the supported service primitives. Two bits for each of the options are used to specify for which service primitives these options are allowed.

- FMA7 Services Supported
This attribute specifies which FMA7 services may be executed on this communication relationship. Two bits for each of the services are used to specify the supported service primitives.

Symbol
This attribute contains the symbolic name of the communication relationship. The length shall be fixed as specified by the attribute Symbol Length in the CRL header.

VFD Pointer
This pointer shall refer to the associated VFD. If a VFD pointer is not supported in the current CRL, as indicated in the CRL header, then the value of this attribute is irrelevant.

User Extension (optional)
This optional attribute contains CREF related configuration information for the FMA7-user. It can only be transfered remotly and contains one or more Parameter-Octets, each in conjunction with 1 or more data octets. The Parameter-Octet is coded as follows:

```
+---------------+
!F!P!P!P!P!P!L!L!
+---------------+
```

F: Flag
This bit shall be set equal „1“ for the first Parameter-Octet of the User Extension and equal „0“ in case of following Parameter-Octets.

P:Parameter Code

These bits contain the Parameter-Code and these bits are coded as follows:
00000: Access Protection
00001: Profile Number
00010...01111: reserved for Profiles
10000...11111: vendor specific

L: Length

These bits contain the number of the following data octets and are coded as follows:
00: lenght is coded in the following octet(4 or more data octets)
01: 1 data octet
10: 2 data octets
11: 3 data octets

The following User Extensions are specified in this specification:

Access Protection

This User Extension contains the Password and the Access Groups which shall be used for the Initiate Service for this CREF. The coding is as follows:
Parameter Code: 00000
Length: 10
data octet 1: Password (Unsigned8)
data octet 2: Access Groups (Bit String,coding see FMS)

Profile Number

This User Extension contains the Profile Number which shall be used for the Initiate Service for this CREF. The coding is as follows:
Parameter Code: 00001
Length: 10
data octet 1 and 2: Profile Number (Octet String, coding see FMS)

Description of the attributes for the dynamic part of a CRL entry

The value of attributes which are not needed for the current communication relationship are irrelevant, but the attributes shall exist.

Status

This attribute contains the status of all LLI state machines of the communication relationship.

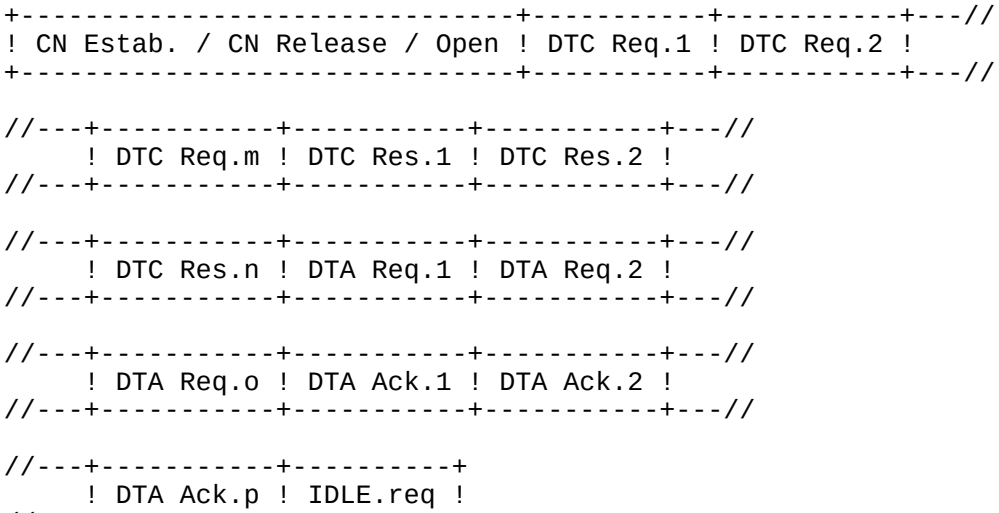


Figure 13. Structure of the Status Attribute on Connections for Acyclic Data Transfer (Master)

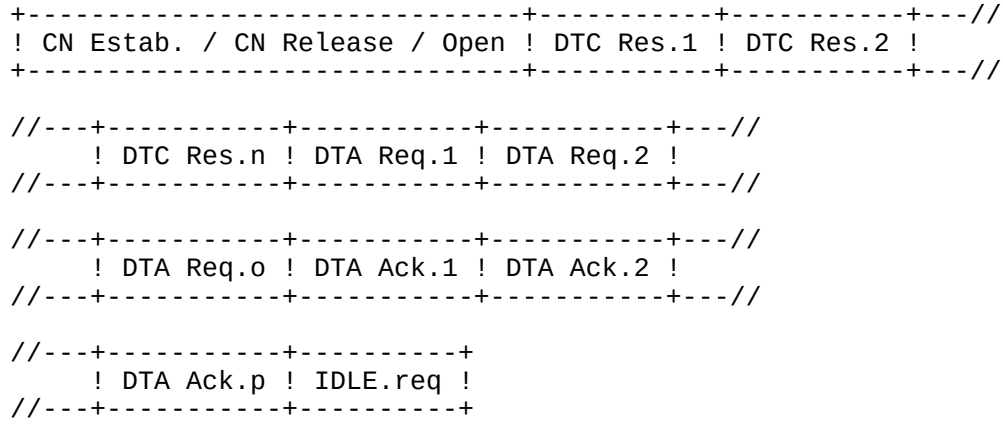


Figure 14. Structure of the Status Attribute on Connections for Acyclic Data Transfer (Slave)

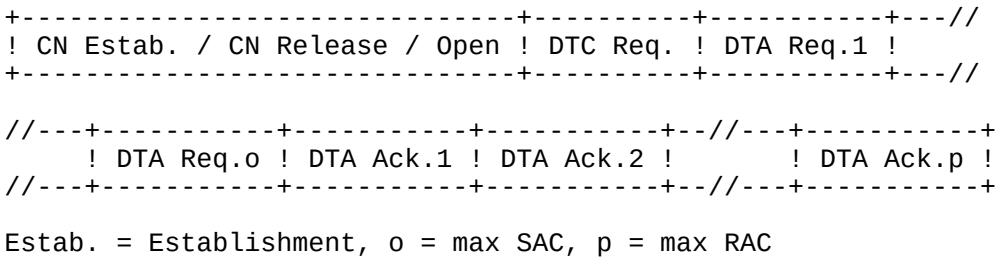


Figure 15. Structure of the Status Attribute on a Connection for Cyclic Data Transfer at the Requester (Master)

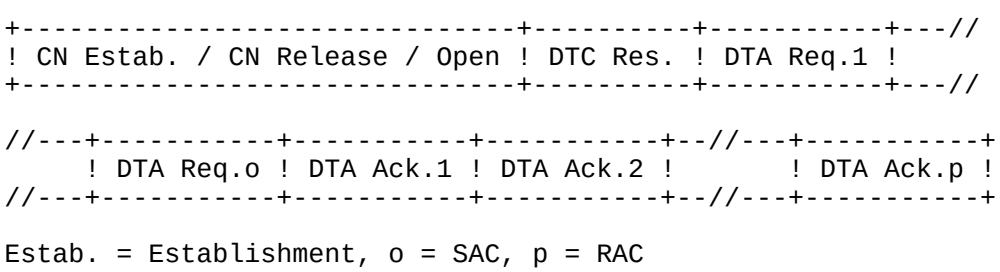


Figure 16. Structure of the Status Attribute on a Connection for Cyclic Data Transfer at the Responder (Slave)

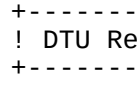


Figure 17. Structure of the Status Attribute of a Broadcast/Multicast Communication Relationship at the Requester

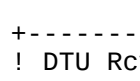


Figure 18. Structure of the Status Attribute of a Broadcast/Multicast Communication Relationship at the Receiver

Valid states:
state machines: CN-Establishment / CN-Abort / Open
values - states
(0) - CLOSED
(1) - ASS-REQ-SAP-ACTIVATE
(2) - ASS-SEND-UPDATE
(3) - ASS-POLL-LIST-ON
(4) - ASS-REQ-WAIT-FOR-CON
(5) - ASS-WAIT-FOR-LLI-RES
(6) - ASS-POLL-LIST-OFF
(7) - ASS-RES-SAP-DEACTIVATE

(8) - ASS-RES-SAP-ACTIVATE
(9) - ASS-WAIT-LOC-RES
(10) - ASS-SEND-RES-PDU
(11) - ASS-REPLY-UPDATE
(12) - ABT-UPDATE
(13) - ABT-POLL-LIST-ON
(14) - ABT-SEND-PDU
(15) - ABT-POLL-LIST-OFF
(16) - ABT-SAP-ACTIVATE
(17) - ABT-SAP-DEACTIVATE
(18) - OPEN
(19) - OP-POLL-LIST-ON
(20) - OP-POLL-LIST-OFF
(21) - ASS-WAIT-FOR-UPDATE-CON
(22) - ABT-WAIT-FOR-UPDATE-CON

state machines: DTC-Req.1 - DTC-Req.m
values - states
(0) - DTC-WAIT-FOR-REQ
(1) - DTC-SEND-UPDATE
(2) - DTC-WAIT-FOR-CON
(3) - DTC-WAIT-FOR-RES

state machines: DTC-Res.1 - DTC-Res.n
values - states
(0) - DTC-WAIT-FOR-REQ_PDU
(1) - DTC-WAIT-FOR-LOC-RES
(2) - DTC-REPLY-UPDATE
(3) - DTC-SEND-RES

state machines: DTA-Req.1 - DTA-Req.o
values - states
(0) - DTA-WAIT-FOR-REQM
(1) - DTA-SEND-UPDATE
(2) - DTA-WAIT-FOR-CON
(3) - DTA-WAIT-FOR-REQS
(4) - DTA-REPLY-UPDATE
(5) - DTA-WAIT-FOR-IND

state machines: DTA-Ack.1 - DTA-Ack.p
values - states
(0) - DTA-WAIT-FOR-REQ-PDU
(1) - DTA-H-WAIT-FOR-BUFFER-FREE
(2) - DTA-H-REPLY-UPDATE
(3) - DTA-H-SEND-ACK
(4) - DTA-L-WAIT-FOR-BUFFER-FREE
(5) - DTA-L-UPDATE
(6) - DTA-L-SEND-ACK

state machine: IDLE.req
values - states
(0) - IDLE-WAIT-FOR-REQM
(1) - IDLE-WAIT-FOR-CON
(2) - IDLE-SEND-UPDATE
(3) - IDLE-WAIT-FOR-REQS

(4) - IDLE-REPLY-UPDATE
(5) - IDLE-WAIT-FOR-IND

state machine: DTU-Req.
values - states
(0) - CONLS-REQUESTER
(1) - CONLS-WAIT-FOR-CON

state machine: DTU-Rcv.
values - states
(0) - CONLS-SERVER

state machine: DTCC-Req:
values - states
(0) - DTCC-WAIT-FOR-REQ
(1) - DTCC-SEND-FOR-REQ
(2) - DTCC-WAIT-FOR-CON
(3) - DTCC-WAIT-FOR-RES
(4) - DTCC-WAIT-FOR-CYC-RES

state machine: DTCC-Res:
values - states
(0) - DTCC-WAIT-FOR-REQ-PDU
(1) - DTCC-WAIT-FOR-FMS-RES
(2) - DTCC-REPLY-UPDATE
(3) - DTCC-SEND-RES

Actual Remote Address
This attribute contains the address of the communication partner which initiated the connection establishment on this communication relationship.

Valid values
(0..126) - station address
(255) - ALL

Actual Remote LSAP
This attribute contains the Service Access Point of the communication partner which initiated the connection establishment on this communication relationship.

Valid values
(0..62) - SAP values
(128) - NIL
(255) - ALL

SCC (Send Confirmed Request Counter)
This attribute specifies the current number of parallel confirmed services to the communication partner.

RCC (Receive Confirmed Request Counter)
This attribute specifies the current number of parallel confirmed services from the communication partner.

SAC (Send Acknowledged Request Counter)
This attribute specifies the current number of parallel unconfirmed services to the communication partner.

RAC (Receive Acknowledged Request Counter)
This attribute specifies the current number of parallel unconfirmed services from the communication partner.

Poll Entry Enabled

This attribute specifies the status of the Poll List entry of this communication relationship. It is equivalent to the attribute Poll Entry of the LLI CRL.

FALSE - The poll list entry in Layer 2 is locked, i.e. the entered station (Rem_add/DSAP) is not polled.

TRUE - The poll list entry is unlocked, i.e. the entered station (Rem_add/DSAP) is polled.

Attributes

Object: Communication Relationship List
  Key Attribute: implicit
  Attribute: CRL Header
  Attribute: List of CRL Entry

Object: CRL Header
  Key Attribute: Communication Reference
  Attribute: Number of CRL Entries
  Attribute: Poll List SSAP
  Attribute: ASS/ABT CI
  Attribute: Symbol Length
  Attribute: VFD Pointer Supported

Object: CRL Entry
  Key Attribute: Communication Reference
  Attribute: Local LSAP
  Attribute: Remote Address
  Attribute: Remote LSAP
  Attribute: Type
  Attribute: LLI SAP
  Attribute: Connection Attribute
  Attribute: max SCC
  Attribute: max RCC
  Attribute: max SAC
  Attribute: max RAC
  Attribute: CI
  Attribute: Multiplier
  Attribute: Max PDU Sending High Prio
  Attribute: Max PDU Sending Low Prio
  Attribute: Max PDU Receiving High Prio
  Attribute: Max PDU Receiving Low Prio
  Attribute: Services Supported
  Attribute: Symbol
  Attribute: VFD Pointer
  Attribute: Status
  Attribute: Actual Remote Address
  Attribute: Actual Remote LSAP
  Attribute: SCC
  Attribute: RCC
  Attribute: SAC
  Attribute: RAC
  Attribute: Poll Entry Enabled

4.5.1.3 CRL Management Services

Operations on the CRL are carried out with the following CRL management services:

- InitiateLoad-CRL-Loc
- Load-CRL-Loc

- TerminateLoad-CRL-Loc
- InitiateLoad-CRL-Rem
- Load-CRL-Rem
- TerminateLoad-CRL-Rem
- Read-CRL-Loc
- Read-CRL-Rem

The CRL header or one CRL entry is read out with the Read-CRL service.

The CRL is written with the Load-CRL service. The write access to the CRL shall be started with the InitiateLoad-CRL service and concluded with the Terminate-Load-CRL service. The CRL entries shall be loaded in the following order:

- CRL Header
- All other CRL entries in ascending order without gaps.

The CRL entry with CREF 1 can be loaded (only locally) if one of the following conditions is true:

- management connection is closed or
- CREF 1 not existing in the stored CRL.

If no CRL entry with CREF 1 is loaded then the stored CRL entry with CREF 1 shall be used.

A stored CRL shall be deleted by loading a CRL Header with the attribute Number of CRL Entries equals 0.

The communication does not test the CRL for consistency but only for ability to be loaded.

The CRL may be read and written locally or remotely. Reading and writing is not permissible at the same time.

4.5.1.3.1 Read-CRL-Loc

The CRL header or one CRL entry is read with the Read-CRL service.

The communication reference shall be set to 0 for reading the CRL header. FMA7 composes the CRL header out of the FMA7, the FMS and the LLI CRL headers.

The corresponding communication reference shall be set for reading a CRL entry. In the case of a management connection, FMA7 composes the CRL entry out of the FMA7 CRL entry and the LLI CRL entry. In the case of a FMS connection it is composed out of the FMS CRL entry and the LLI CRL entry. If this CREF does not exist, the response to the service is Result(-). The Read-CRL-Loc service shall be used repeatedly to read out the whole CRL.

Table 4. Read-CRL-Loc

Parameter Name	.req	.con
Argument	M	
Desired CREF	M	
Result(+)		S
CRL Entry		M
Result(-)		S
Error Type		M

Argument
The argument contains the parameters of the Read-CRL-Loc.req primitive.

Desired CREF

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

CRL Entry
CRL header or one CRL entry.

Result(-)
The Result(-) parameter indicates that the service failed.

Error Type
This parameter contains information on the reason the service failed.

4.5.1.3.2 Read-CRL-Rem

The CRL header or one CRL entry is read at the remote station with the Read-CRL-Rem service. For reading the CRL header the communication reference shall be set to 0. For reading the CRL entry the corresponding communication reference shall be set. If this CREF does not exist the response to the service is Result(-). The Read-CRL-Rem service shall be used repeatedly to read out the whole CRL.

Table 5. Read-CRL-Rem

Parameter Name	.req	.ind	.res	.con
Argument	M			
Communication Reference	M			
Desired CREF	M			
Result(+)			S	
Communication Reference			M	
CRL Entry			M	
Result(-)			S	
Communication Reference			M	
Error Type			M	

Argument
The argument contains the parameters of the Read-CRL-Rem.req primitive and the Read-CRL-Rem.ind primitive.

Communication Reference
This parameter specifies the identifier of the associated management connection in the CRL.

Desired CREF
The value 0 is set for reading the CRL header, otherwise the communication reference of the CRL entry which is to be read.

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

CRL Entry
CRL header or one CRL entry.

Result(-)
The Result(-) parameter indicates that the service failed.

Error Type
This parameter contains information on the reason the service failed.

4.5.1.3.3 InitiateLoad-CRL-Loc

The local writing into the CRL is performed with a sequence of InitiateLoad-CRL-Loc, one or several Load-CRL-Loc services and one TerminateLoad-CRL-Loc.

All communication relationships with the exception of the default management connection are released by FMS and LLI, and stay locked for data communication until the Terminate-Load-CRL-Loc service was executed successfully. After the execution of the InitiateLoad-CRL-Loc service the CRL header and the CRL entries may be loaded with the Load-CRL-Loc service.

Table 6. InitiateLoad-CRL-Loc

Parameter Name	.req	.con
Argument	M	
Result(+)		S
Result(-)		S
Error Type		M

Argument
The argument contains no parameters.

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

Result(-)
The Result(-) parameter indicates that the service failed.
Error Type: information on the reason the service failed.

4.5.1.3.4 Load-CRL-Loc

The CRL header or the static part of one CRL entry is written locally with the Load-CRL-Loc service. The CRL header is written as a CRL entry with the communication reference 0. FMA7 generates out of the CRL header the FMA7, FMS and the LLI CRL headers and writes these into the corresponding instances.

For writing the static part of a CRL entry the corresponding CREF shall be set. The FMA7 generates out of the CRL entry the FMA7 CRL entry and the LLI CRL entry in the case of a management connection. In the case of a FMS connection it generates the FMS CRL entry and the LLI CRL entry. Then the CRL entries are written into the corresponding instances.

The Load-CRL-Loc service shall be used repeatedly to write the whole CRL.

Table 7. Load-CRL-Loc

Parameter Name	.req	.con
Argument	M	
CRL Entry Static	M	
Result(+)		S
Result(-)		S
Error Type		M

Argument
The argument contains the parameters of the Load-CRL-Loc.req primitive.

CRL Entry Static
CRL header or static part of a CRL entry.

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

Result(-)
The Result(-) parameter indicates that the service failed.

Error Type
This parameter contains information on the reason the service failed.

4.5.1.3.5 TerminateLoad-CRL-Loc

The TerminateLoad-CRL-Loc service concludes the local writing into the CRL. FMS and LLI are unlocked again for data transfer.

Table 8. TerminateLoad-CRL-Loc

Parameter Name	.req	.con
Argument	M	
Result(+)		S
Result(-)		S
Error Type		M
Error CREF		C

Argument
The argument contains no parameters.

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

Result(-)
The Result(-) parameter indicates that the service failed. If an error occurred

during unlocking of FMS or LLI, then all communication relationships remain released with the exception of the management connection.

Error Type
This parameter contains information on the reason the service failed.

Error CREF
This parameter specifies at which CREF the TerminateLoad-CRL-Loc service was aborted. The existence of the parameter depends on the Error Type.

4.5.1.3.6 InitiateLoad-CRL-Rem

The remote writing into the CRL is done with a service sequence of InitiateLoad-CRL-Rem, one or several Load-CRL-Rem and one TerminateLoad-CRL-Rem.

Table 9. InitiateLoad-CRL-Rem

Parameter Name	.req .ind	.res .con
Argument	M	
Communication Reference	M	
Result(+)		S
Communication Reference		M
Result(-)		S
Communication Reference		M
Error Type		M

Argument
The argument contains the parameters of the InitiateLoad-CRL-Rem.req primitive and the InitiateLoad-CRL-Rem.ind primitive.

Communication Reference
This parameter specifies the identifier of the associated management connection in the CRL.

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

Result(-)
The Result(-) parameter indicates that the service failed.

Error Type
This parameter contains information on the reason the service failed.

4.5.1.3.7 Load-CRL-Rem

The CRL header or the static part of a CRL entry is written remotely with the Load-CRL-Rem service. The CRL header is written as a CRL entry with the communication reference 0. For writing the static part of a CRL entry the corresponding CREF shall be set. The Load-CRL-Rem service shall be used repeatedly to write the whole CRL.

Table 10. Load-CRL-Rem

Parameter Name	.req .ind	.res .con
Argument	M	
Communication Reference	M	
CRL Entry Static	M	
Result(+)		S
Communication Reference		M
Result(-)		S
Communication Reference		M
Error Type		M

Argument
The argument contains the parameters of the Load-CRL-Rem.req primitive and the Load-CRL-Rem.ind primitive.

Communication Reference
This parameter specifies the identifier of the associated management connection in the CRL.

CRL Entry Static
CRL header or static part of the CRL entry.

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

Result(-)
The Result(-) parameter indicates that the service failed.

Error Type
This parameter contains information on the reason the service failed.

4.5.1.3.8 TerminateLoad-CRL-Rem

The TerminateLoad-CRL-Rem service concludes the remote writing of the CRL.

Table 11. TerminateLoad-CRL-Rem

Parameter Name	.req .ind	.res .con
Argument	M	
Communication Reference	M	
Result(+)		S
Communication Reference		M
Result(-)		S
Communication Reference		M
Error Type		M
Error CREF		C

Argument
The argument contains the parameters of the TerminateLoad-CRL-Rem.req primitive and the TerminateLoad-CRL-Rem.ind primitive.

Communication Reference
This parameter specifies the identifier of the associated management connection in the CRL.

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

Result(-)
The Result(-) parameter indicates that the service failed. If an error occurred during unlocking the FMS or LLI, then all communication relationships remain released with the exception of the management connection.

Error Type
This parameter contains information on the reason the service failed.

Error CREF
This parameter specifies at which CREF the TerminateLoad-CRL-Rem service was aborted. The existence of the parameter depends on the Error Type.

4.5.1.4 State Machine for Read-CRL

State Machine Description

The state machine for reading the CRL has 3 states.

CRL-UPLOAD-READY

CRL-Upload is ready.

FMS-UPLOAD
One or several CRL entries are being read out of the FMS CRL.

LLI-UPLOAD
One or several CRL entries are being read out of the LLI CRL.

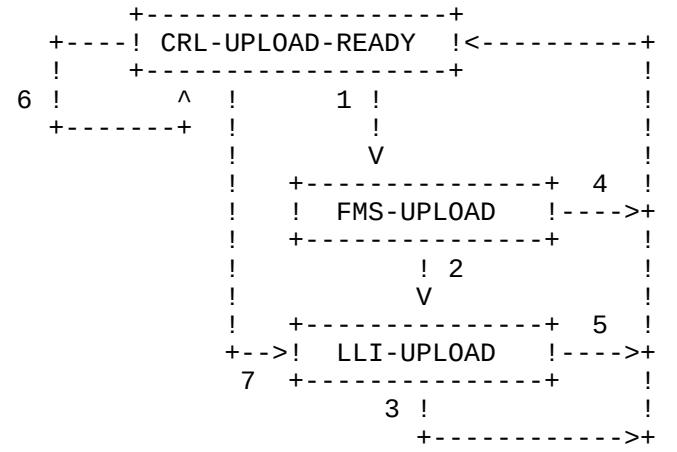


Figure 19. State Machine for Reading the CRL

State Transitions

Read of the CRL

Current State / Event / \Exit Condition / => Action Taken	**Transition**	**Next State**
CRL-UPLOAD-READY Read-CRL-Loc.req \CRL-LOADER-STATUS = CRL-LOADER-READY AND (desired CREF not present in FMA7 CRL OR desired CREF = 0) => FMS Read-CRL.req	**1**	**FMS-UPLOAD**
FMS-UPLOAD FMS Read-CRL.con(+) => LLI Read-CRL.req < CREF from FMS Read-CRL.con(+) >	**2**	**LLI-UPLOAD**
LLI-UPLOAD LLI Read-CRL.con(+) => Read-CRL-Loc.con(+)	**3**	**CRL-UPLOAD-READY**
FMS-UPLOAD FMS Read-CRL.con(-) => Read-CRL-Loc.con(-)	**4**	**CRL-UPLOAD-READY**
LLI-UPLOAD LLI Read-CRL.con(-) => Read-CRL-Loc.con(-)	**5**	**CRL-UPLOAD-READY**
CRL-UPLOAD-READY Read-CRL-Loc.req \CRL-LOADER-STATUS = CRL-LOADER-READY => Read-CRL-Loc.con(-)	**6**	**CRL-UPLOAD-READY**
CRL-UPLOAD-READY Read-CRL-Loc.req \CRL-LOADER-STATUS = CRL-LOADER-READY AND desired CREF present in FMA7 CRL AND desired CREF <> 0 => read FMA7 CRL entry out of FMA7 CRL LLI Read-CRL.req	**7**	**LLI-UPLOAD**

4.5.1.5 State Machine for Load-CRL

The state machine for loading the CRL has 9 states.

State Machine Description

CRL-LOADER-READY
The CRL loader is ready.

FMS-DISABLE-1
The FMS releases all communication relationships with the exception of the management connection.

LLI-DISABLE
The LLI releases all communication relationships with the exception of the management connection.

CRL-LOADER-ACTIVE
The CRL loader is in the loading state.

FMS-LOADING
One or several CRL entries are being loaded into the FMS CRL.

LLI-LOADING
One or several CRL entries are being loaded into the LLI CRL.

FMS-ENABLE
The FMS is unlocked.

LLI-ENABLE
The LLI is unlocked.

FMS-DISABLE-2
The FMS releases all communication relationships with the exception of the management connection.

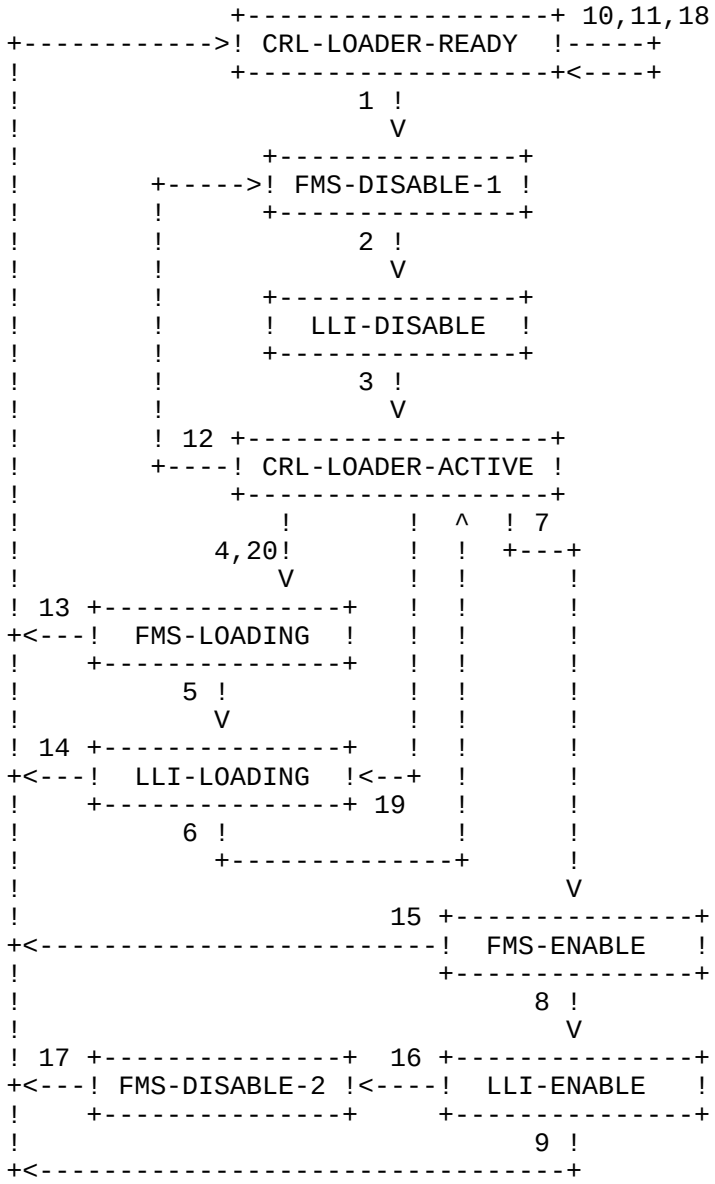


Figure 20. State Machine for Loading the CRL

State Transitions

Loading of the CRL

Current State Event \Exit Condition => Action Taken	**Transition**	**Next State**
CRL-LOADER-READY InitiateLoad-CRL-Loc.req \CRL upload status = CRL-UPLOAD-READY => FMS Disable.req release all established connections and currently establishing connections, except the default management connection, <RC = ABT_RC12>	**1**	**FMS-DISABLE-1**
CRL-LOADER-READY Load-CRL-Loc.req => Load-CRL-Loc.con(SC)	**10**	**CRL-LOADER-READY**
CRL-LOADER-READY TerminateLoad-CRL-Loc.req => TerminateLoad-CRL-Loc.con(SC)	**11**	**CRL-LOADER-READY**
FMS-DISABLE-1 FMS Disable.con => LLI Disable.req	**2**	**LLI-DISABLE**
LLI-DISABLE LLI Disable.con => InitiateLoad-CRL-Loc.con(+)	**3**	**CRL-LOADER-ACTIVE**
CRL-LOADER-ACTIVE Load-CRL-Loc.req \ CRL entry: CREF = 0 => generate FMA7 CRL header FMS Load-CRL.req	**4**	**FMS-LOADING**
CRL-LOADER-ACTIVE Load-CRL-Loc.req \ CRL entry: LLI SAP = 0 => FMS Load-CRL.req	**20**	**FMS-LOADING**
CRL-LOADER-ACTIVE TerminateLoad-CRL-Loc.req => FMS Enable.req	**7**	**FMS-ENABLE**
CRL-LOADER-ACTIVE InitiateLoad-CRL-Loc.req => FMS-Disable.req	**12**	**FMS-DISABLE-1**
FMS-LOADING FMS Load-CRL.con(+) => LLI Load-CRL.req	**5**	**LLI-LOADING**
FMS-LOADING FMS Load-CRL.con(-) => Load-CRL-Loc.con(-)	**13**	**CRL-LOADER-READY**

Loading of the CRL

Current State Event \Exit Condition => Action Taken	Transition	Next State
LLI-LOADING LLI Load-CRL.con(+) => Load-CRL-Loc.con(+)	**6**	**RL-LOADER-ACTIVE**
LLI-LOADING LLI Load-CRL.con(-) => Load-CRL-Loc.con(-)	**14**	**CRL-LOADER-READY**
FMS-ENABLE FMS Enable.con(+) => LLI Enable.req	**8**	**LLI-ENABLE**
FMS-ENABLE FMS Enable.con(-) => TerminateLoad-CRL-Loc.con(-)	**15**	**CRL-LOADER-READY**
LLI-ENABLE LLI Enable.con(+) => TerminateLoad-CRL-Loc.con(+) pre-set dynamic part of the FMA7 CRL	**9**	**CRL-LOADER-READY**
LLI-ENABLE LLI Enable.con(-) => FMS Disable.req	**16**	**FMS-DISABLE-2**
FMS-DISABLE-2 FMS Disable.con => TerminateLoad-CRL-Loc.con(-)	**17**	**CRL-LOADER-READY**
CRL-LOADER-READY InitiateLoad-CRL-Loc.req \CRL-UPLOAD-STATUS = CRL-UPLOAD-READY => InitiateLoad-CRL-Loc.con(SC)	**18**	**CRL-LOADER-READY**
CRL-LOADER-ACTIVE Load-CRL-Loc.req \ CRL entry: LLI SAP = 1 => CRL entry static in FMA7 CRL LLI Load-CRL.req	**19**	**LLI-LOADING**

4.5.2 FDL Service Access Point

The data transfer services are executed over Service Access Points (LSAP, Link Service Access Point) at the interface LLI/FDL.

4.5.2.1 Model Description

Services are provided to the FMA7 user to obtain information about the configuration of Service Access Points of the Layer 2 with regard to their FDL services at the local or the remote station.

4.5.2.2 The LSAP Object

The LSAP object is in Layer 2 and is described there.

Attributes

Object: LSAP
  Key Attribute: LSAP value
  Attribute: Access
  Attribute: List of Activate Service
    Attribute: Service_activate
    Attribute: Role_in_service

The range of values is specified in Layers 1/2, if not given here.

LSAP value
Uniquely addresses the LSAP object.

Valid values
(0...63) - SAP values
(128)   - NIL

Access
This attribute defines whether an access protection exists for this LSAP (ALL/Rem_Add).

List of Activate Service
This attribute contains the list of the attributes Service_activate and Role_in_service.

Service_activate
This attribute defines which FDL services are activated for this LSAP (SDA, SDN, SRD, CSRD).

Role_in_service
This attribute specifies the configuration for the activated service (Initiator, Responder, Both).

4.5.2.3 Services

4.5.2.3.1 LSAP-StatusLoc

The FMA7 user may request the configuration of a local Service Access Point with the LSAP-StatusLoc service.

Table 12. LSAP-StatusLoc

Parameter Name	.req	.con
Argument	M	
LSAP Value	M	
Result(+)		S
LSAP_Status_Data_Unit		M
Result(-)		S
Error Type		M

Argument
The argument contains the parameters of the LSAP-StatusLoc.req primitive.

LSAP Value
This parameter specifies the value of the local Service Access Point.

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

LSAP_Status_Data_Unit
This parameter contains the LSAP_Status_Data_Unit of Layer 2.

Result(-)
The Result(-) parameter indicates that the service failed.

Error Type
This parameter contains information on the reason the service failed. If the LSAP is not activated the Error Type:
access/object-non-existent shall be returned

4.5.2.3.2 LSAP-StatusRem

The FMA7 user may request the configuration of a Service Access Point at a remote station with the LSAP-StatusRem service.

Table 13. LSAP-StatusRem

Parameter Name	.req / .ind	.res / .con
Argument	M	
Communication Reference	M	
LSAP Value	M	
Result(+)		S
Communication Reference		M
LSAP_Status_Data_Unit		M
Result(-)		S
Communication Reference		M
Error Type		M

Argument
The argument contains the parameters of the LSAP-StatusRem.req primitive and the LSAP-StatusRem.ind primitive.

Communication Reference
This parameter specifies the identifier of the associated management connection in the CRL.

LSAP Value
This parameter specifies the value of the Service Access Point at the remote station.

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

LSAP_Status_Data_Unit
This parameter contains the LSAP_Status_Data_Unit of Layer 2.

Result(-)
The Result(-) parameter indicates that the service failed.

Error Type
This parameter contains information on the reason the service failed.

4.5.3 PHY/FDL Variables

The PHY/FDL variables are operational parameters of the Layers 1/2 and counters of Layer 2. The meaning and the range of values of the variables and the counters shall be found in part 2.

4.5.3.1 Model Description

Services are provided to the FMA7 user to read or to write the PHY/FDL variables locally or remotely.

4.5.3.2 Attributes

Object: PHY/FDL Variable
  Key Attribute: Variable Identifier
  Attribute: Value

Variable Identifier
Identifier for the variable. The variable identifier specifies implicitly to which instance the variable is assigned.

Value
Value of the variable.

4.5.3.3 Services for PHY/FDL Variables

SetValueLoc

Local variables in FDL or PHY are set with the SetValueLoc service.

Table 14. SetValueLoc

Parameter Name	.req	.con
Argument	M	
Variable Identifier	M	
Desired Value	M	
Result(+)		S
Result(-)		S
Error Type		M

Argument
The argument contains the parameters of the SetValueLoc.req primitive.

Variable Identifier
This parameter specifies the variable which to receive a new value. Valid values:

FDL operational parameters
(1) - TS
(2) - Baud_rate
(3) - Medium_red

(4) - HW-Release
(5) - SW-Release
(6) - TSL
(7) - MIN_TSDR
(8) - MAX_TSDR
(9) - TQUI
(10) - TSET
(11) - TTR
(12) - G
(13) - In_ring_desired
(14) - HSA
(15) - max_retry_limit

counters
(20) - Frame_sent_count
(21) - Retry_count
(22) - SD_count
(23) - SD_error_count

PHY variables
(30) - Transmitter_output
(31) - Received_signal_source
(32) - Loop

Desired Value
This parameter contains the value of the selected variable. The valid range of values is defined in the subclauses 5.4.3.4 and 5.4.3.5.

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

Result(-)
The Result(-) parameter indicates that the service failed.

Error Type
This parameter contains information on the reason the service failed.

4.5.3.3.1 ReadValueLoc

Local variables are read out of FDL or PHY with the ReadValueLoc service.

Table 15. ReadValueLoc

Parameter Name	.req	.con
Argument	M	
Variable Identifier	M	
Result(+)		S
Current Value		M
Result(-)		S
Error Type		M

Argument
The argument contains the parameters of the ReadValueLoc.req primitive.

Variable Identifier
This parameter specifies the variable whose value is to be read.

Valid values
FDL operational parameters:
(1) - TS
(2) - Baud_rate
(3) - Medium_red
(4) - HW-Release
(5) - SW-Release
(6) - TSL
(7) - MIN_TSDR
(8) - MAX_TSDR
(9) - TQUI
(10) - TSET
(11) - TTR
(12) - G
(13) - In_ring_desired
(14) - HSA
(15) - max_retry_limit
(16) - TRR
(17) - LAS
(18) - GAPL

counters
(20) - Frame_sent_count
(21) - Retry_count
(22) - SD_count
(23) - SD_error_count

PHY variables
(30) - Transmitter_output
(31) - Received_signal_source
(32) - Loop

Result(+)

The Result(+) parameter indicates that the service was executed successfully.

Current Value
This parameter contains the value of the selected variable. The valid range of values is defined in PHY/FDL Variables specification.

Result(-)
The Result(-) parameter indicates that the service failed.

Error Type
This parameter contains information on the reason the service failed.

4.5.3.3.2 SetValueRem

Variables are set in the FDL of the remote station with the SetValueRem service. Only the variables which are not critical for the stability of the token ring may be manipulated.

Table 16. SetValueRem

Parameter Name	.req .ind	.res .con
Argument	M	
Communication Reference	M	
Variable Identifier	M	
Desired Value	M	
Result(+)		S
Communication Reference		M
Result(-)		S
Communication Reference		M
Error Type		M

Argument
The argument contains the parameters of the SetValueRem.req primitive and the SetValueRem.ind primitive.

Communication Reference
This parameter specifies the identifier of the associated management connection in the CRL.

Variable Identifier
This parameter specifies the variable which is to receive a new value.

Valid values

FDL operational parameters
(3) - Medium_red
(6) - TSL
(7) - MIN_TSDR
(8) - MAX_TSDR
(9) - TQUI
(10) - TSET
(11) - TTR
(12) - G
(13) - In_ring_desired
(14) - HSA
(15) - max_retry_limit

counters
(20) - Frame_sent_count
(21) - Retry_count
(22) - SD_count
(23) - SD_error_count

Desired Value
This parameter contains the value of the selected variable. The valid range of values is defined in the subclause 5.4.3.4.

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

Result(-)
The Result(-) parameter indicates that the service failed.

Error Type
This parameter contains information on the reason the service failed.

4.5.3.3.3 ReadValueRem

Variables are read out of FDL or PHY of the remote station with the ReadValueRem service.

Table 17. ReadValueRem

Parameter Name	.req .ind	.res .con
Argument	M	
Communication Reference	M	
Variable Identifier	M	
Result(+)		S
Communication Reference		M
Current Value		M
Result(-)		S
Communication Reference		M
Error Type		M

Argument
The argument contains the parameters of the ReadValueRem.req primitive and of the ReadValueRem.ind primitive.

Communication Reference
This parameter specifies the identifier of the associated management connection in the CRL.

Variable Identifier
This parameter specifies the variable whose value is to be read.

Valid values
FDL operational parameters:
(1) - TS
(2) - Baud_rate
(3) - Medium_red
(4) - HW-Release
(5) - SW-Release
(6) - TSL
(7) - MIN_TSDR
(8) - MAX_TSDR
(9) - TQUI
(10) - TSET
(11) - TTR
(12) - G
(13) - In_ring_desired

(14) - HSA
(15) - max_retry_limit
(16) - TRR
(17) - LAS
(18) - GAPL

counters
(20) - Frame_sent_count
(21) - Retry_count
(22) - SD_count
(23) - SD_error_count

PHY variables
(30) - Transmitter_output
(31) - Received_signal_source
(32) - Loop

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

Current Value

This parameter contains the value of the selected variable. The valid range of values is defined in the subclauses PHY/FDL Variables specification.

Result(-)
The Result(-) parameter indicates that the service failed.

Error Type
This parameter contains information on the reason the service failed.

4.5.3.4 Range of Values of FDL Variables

The range of values of the FDL variables and counters is specified in the Data Link Layer of the PROFIBUS Specification.

This subclause contains additional definitions for all variables for which an exact encoding is not specified by the definition of the data type in FMA7 and the definitions in the Data Link Layer of the PROFIBUS Specification.

Baud_rate
(0) - 9,6 kbit/s
(1) - 19,2 kbit/s
(2) - 93,75 kbit/s
(3) - 187,5 kbit/s
(4) - 500 kbit/s
(5) - 38,4 kbit/s
(6) - 1500 kbit/s

Medium_red
(0) - no_redundancy
(1) - bus_A_highprior
(2) - bus_B_highprior

LAS
Unsigned8 - Length of LAS (m-n+1)
Unsigned8 - FDL Address Active Station n
Unsigned8 - FDL Address Actice Station n+1
...
Unsigned8 - FDL Address Active Station n+m

The length of LAS is defined as the number of octets following the length field.

GAPL
Unsigned8 - Length of GAPL (m-n+2)
Unsigned8 - Start FDL Address
Unsigned8 - HSA
Unsigned8 - FDL_Status n
Unsigned8 - FDL Status n+1
...
Unsigned8 - FDL Status n+m

FDL Status
(0) - passive station at this address
(1) - active station not ready at this address
(2) - station ready for token ring
(3) - active station in token ring
(4) - no station available at this address

The length of GAPL is defined as the number of octets following the length field.
If there is no GAP then the Start FDL Address is irrelevant and the HSA field shall contain the current HSA value. The information „no GAP“ shall be indicated by the value of 2 of length field.

In_ring_desired:
TRUE - the station may be taken into the logical token ring
FALSE - the station may not be taken into the logical token ring

4.5.3.5 Range of Values of PHY Variables

Transmitter_output
TRUE - enabled
FALSE - disabled

Received_signal_source
(0) - primary
(1) - alternate
(2) - random

Loop
TRUE - enabled
FALSE - disabled

4.5.4 Identification

The FMA7 identification services allow the FMA7 user to request the manufacturer, the software and hardware releases and the characteristics of a PROFIBUS controller. In contrast to the FMS identification service, the communication components are identified here.

4.5.4.1 Model Description

Services for reading the identification reports (Ident List) of individual instances (FMS, LLI, FDL, FMA7) and for reading the identification report (Station Ident List) of the complete PROFIBUS communication system are provided to the user. FMS, LLI, FDL and FMA7 each have an object Ident List. FMA7 has in addition an object Ident List which describes the specific realization of the complete PROFIBUS communication system. The lengths of an Ident List and of the Station Ident list may not exceed 200 octets.

4.5.4.2 The Ident List Object

The object is defined in FMA7, FMS, LLI and FDL.

Attributes

```
Object: Ident List
  Key Attribute: implicit by Instance Identifier
  Attribute: Vendor Name
  Attribute: Controller Type
  Attribute: Hardware Release
  Attribute: Software Release
```

Instance Identifier
This attribute specifies from which instance the Ident List is requested.

Valid values
- FMA7
- FMS
- LLI
- FDL

Vendor Name
This attribute specifies the vendor of the software of the associated instance.

Controller Type
This attribute specifies the hardware (e.g. processor, controller, ASIC), on which this instance is implemented.

Hardware Release
This attribute specifies the hardware release of the associated instance.

Software Release
This attribute specifies the software release of the associated instance.

4.5.4.3 Station Ident List Object

This object contains the identification report and the communication specific features of the PROFIBUS communication system in the given realization.

Attributes

```
Object: Station Ident List
  Key Attribute: implicit by Instance Identifier
  Attribute: Vendor Name
  Attribute: Controller Type
  Attribute: Hardware Release
  Attribute: Software Release
  Attribute: Characteristics
```

Instance Identifier
This attribute specifies from which instance the Ident List is requested.

Valid values
- Station

Vendor Name
This attribute specifies the vendor of the PROFIBUS communication system.

Controller Type
This attribute specifies the hardware of the PROFIBUS communication system (e.g. controller, board).

Hardware Release
This attribute specifies the hardware release of the PROFIBUS communication system.

Software Release
This attribute specifies the software release of the PROFIBUS communication system.

Characteristics
This attribute specifies the features of the specific realization of the PROFIBUS communication system.

Byte	Data Type	Meaning
1-3	Bit String	Functions Supported
4-9	Bit String	FMS Features Supported (see table 13)
10-15	Bit String	FMA7 Services Supported (see table 112)
16	Unsigned8	Highest allowed SAP Value (< 63)
17	Unsigned8	Maximum number of SAPs
18-19	Unsigned16	Highest allowed CREF
20-21	Unsigned16	Maximum number of CRL entries
22-25	Unsigned32	Total length of all PDU Buffers
26-27	Unsigned16	Total number of parallel services
28	Unsigned8	Maximum PDU length
29-30	Unsigned16	Largest Index in the OD
31-32	Unsigned16	Maximum number of OD Entries
33	Unsigned8	Maximum number of VFDs
34	Unsigned8	Maximum Entries in the LAS
35	Unsigned8	MIN-TSDR-ABS **1)
36-37	Octet String	Profile Number
38	Unsigned8	TRDY-ABS **1)
39	Unsigned8	TSDI-ABS **1)
40	Unsigned8	MAX-TSDR-ABS **1)
41-44	Bit String	data types supported
45-47	Bit String	access rights supported
48	Bit String	variable types supported
49-50	Bit String	special functions
51	Unsigned8	max symbol length OD
52	Unsigned8	max symbol length CRL
53	Unsigned8	Tset-ABS **1)
54	Unsigned8	Tout-ABS **1)

**1): absolute value, base is 20 micro seconds

Figure 21. Structure of the Characteristics Attribute

Functions Supported is a bit string of length 24 bits. One bit is used for each of the supported functions.

Bit	Functions supported
0	Master Function in Master/Master CREL
1	Master Function in Master/Slave CREL
2	Slave Function in Master/Slave CREL
3	Connections for Cyclic Data Transfer
4	Connectionless CREL
5	Unconfirmed Services with high priority
6	Named Addressing for Domain
7	Named Addressing for Program Invocation
8	Named Addressing for Simple Variable, Array, Record
9	Named Addressing for Variable List
10	Named Addressing for Event
11	Permanent OD, CRL, Layer 2 parameters
12	9,6 kbit/s permissible
13	19,2 kbit/s permissible
14	93,75 kbit/s permissible
15	187,5 kbit/s permissible
16	500 kbit/s permissible
17	38,4 kbit/s permissible
18	1500 kbit/s permissible
19-23	reserved

Figure 22. Structure of the Bit String Functions Supported

Data Types Supported is a bit string of length 32 bits. One bit is used for each of the Data Types Supported. Reserved bits shall all be „0“.

Bit	Data Types Supported
0	Boolean
1	Integer8
2	Integer16
3	Integer32
4	Unsigned8
5	Unsigned16
6	Unsigned32
7	Floating Point
8	Visible String
9	Octet String
10	Date
11	Time of Day
12	Time Difference
13	Bit String
14-31	reserved

Figure 23. Structure of the Bit String Data Types Supported

Access Rights Supported is a bit string of length 24 bits. One bit is used for each of the Access Rights.

Bit	Access Rights Supported
0	Read all
1	Read with password
2	Read with access group
3	Write all
4	Write with password
5	Write with access group
6	Delete all
7	Delete with password
8	Delete with access group
9	Use all
10	Use with password
11	Use with access group
12	Start all
13	Start with password
14	Start with access group
15	Stop all
16	Stop with password
17	Stop with access group
18	Acknowledge all
19	Acknowledge with password
20	Acknowledge with access group
21	Enable/Disable all
22	Enable/Disable with password
23	Enable/Disable with access group

Figure 24. Structure of the Bit String Access Rights Supported

Variable Types Supported is a bit string of length 8 bits. One bit is used for each of the Variable Types. Reserved bits shall all be „0“.

Bit	Variable Types Supported
0	Simple Variable
1	Array
2	Record
3	Varible List
4-7	reserved

Figure 25. Structure of the Bit String Variable Types Supported

Special Functions Supported is a bit string of length 16 bits. One bit is used for each of the Special Functions. Reserved bits shall all be „0“.

Bit	Special Functions Supported
0	Redundency
1	Unconfirmed services (high prior) as receiver only
2	Sub Index supported as server
3-15	reserved

Figure 26. Structure of the Bit String Variable Types Supported

4.5.4.4 Services

The following services for reading the Ident List or the Station Ident List are provided to the FMA7 user.

- IdentLoc
- IdentRem

4.5.4.4.1 IdentLoc

The Ident List of FMS, FMA7, LLI or FDL, or the Station Ident List is locally read with the IdentLoc service.

Table 18. IdentLoc

Parameter Name	.req	.con
Argument	M	
Instance Identifier	M	
Result(+)		S
Vendor Name		M
Controller Type		M
Hardware Release		M
Software Release		M
Characteristics		C
Result(-)		S
Error Type		M

Argument
The argument contains the parameters of the IdentLoc.req primitive.

Instance Identifier
This parameter indicates from which instance the Ident List or the Station Ident List is to be read.

Valid values
- FMA7
- FMS
- LLI
- FDL
- Station

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

Vendor Name
This parameter contains the vendor name of the associated instance.

Controller Type
This parameter specifies the hardware of the associated instance.

Hardware Release
This parameter specifies the hardware release of the associated instance.

Software Release
This parameter specifies the software release of the associated instance.

Characteristics
This parameter specifies the features of the specific implementation of the PROFIBUS communication system. This parameter is only present if the Instance Identifier has the value Station.

Result(-)
The Result(-) parameter indicates that the service failed.

Error Type
This parameter contains information on the reason the service failed.

4.5.4.4.2 IdentRem

The Ident List of FMS, FMA7, LLI or FDL, or the Station Ident List is read from a remote station with the IdentRem service.

Table 19. IdentRem

Parameter Name	.req .ind	.res .con
Argument	M	
Communication Reference	M	
Instance Identifier	M	
Result(+)		S
Communication Reference		M
Vendor Name		M
Controller Type		M
Hardware Release		M
Software Release		M
Characteristics		C
Result(-)		S
Communication Reference		M
Error Type		M

Argument
The argument contains the parameters of the IdentRem.req primitive and of the IdentRem.ind primitive.

Communication Reference
This parameter specifies the identifier of the associated management connection in the CRL.

Instance Identifier
This parameter indicates from which instance the Ident List or the Station Ident List is to be read.

Valid values
- FMA7
- FMS
- LLI
- FDL
- Station

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

Vendor Name
This parameter contains the vendor name of the associated instance.

Controller Type
This parameter specifies the hardware of the associated instance.

Hardware Release
This parameter specifies the hardware release of the associated instance.

Software Release
This parameter specifies the software release of the associated instance.

Characteristics
This parameter specifies the features of the specific implementation of the PROFIBUS communication system. This parameter is only present if the Instance Identifier has the value Station.

Result(-)
The Result(-) parameter indicates that the service failed.

Error Type
This parameter contains information on the reason the service failed.

4.5.5 Request FDL Status of all Stations (Live List)

4.5.5.1 Model Description

A service for generating the current station list (Live List) is provided to the FMA7 user.

4.5.5.2 The Live List Object

The Live List object is generated by Layer 2 and is delivered to FMA7. The structure of the Live List shall be found in the Data Link Layer of the PROFIBUS Specification.

4.5.5.3 Attributes

```
Object: Live List
  Key Attribute: implicit by the service
  Attribute: List of Live List Entry
    Attribute: FDL Address
    Attribute: Station Type (FDL State)
```

implicit by the service
The Live List is implicitly addressed by the service.

Live List Entry
This attribute contains the FDL address and the station type.

FDL Address
This attribute specifies the FDL address of a currently present and operational station.

Station Type
This attribute specifies the station type (active, passive) and at the active station the FDL status (not ready, ready for the logical token ring, in the logical token ring).

4.5.5.4 Services

4.5.5.4.1 GetLiveList

The user may request a list of all stations currently able to communicate on the bus with the GetLiveList service.

Table 20. GetLiveList

Parameter Name	.req	.con
Argument	M	
Result(+)		S
List of Live List Entries		M
Result(-)		S
Error Type		M

Argument
The argument contains no parameters.

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

List of Live List Entry
This parameter contains the list of stations. The structure and the meaning of the entries shall be found in the Data Link Layer of the PROFIBUS Specification.

Result(-)
The Result(-) parameter indicates that the service failed.

Error Type
This parameter contains information on the reason the service failed.

4.6 Fault Management

The fault management defines local services to reset a PROFIBUS station and to indicate events.

4.6.1 Reset

4.6.1.1 Model Description

A service to reset the communication components (FMS, LLI, FDL, PHY, FMA1/2 and FMA7) is provided to the FMA7 user.

4.6.1.2 Services

The FMA7 Reset service is mandatory.

4.6.1.2.1 FMA7 Reset

With the FMA7 Reset service the complete communication, i.e. FMA7, FMS, LLI, FMA1/2, FDL and PHY, is reset. The restart after a reset is performed in the same way as after a POWER ON. FMA7 transfers a Reset.req primitive to FMA1/2, LLI and FMS after obtaining a FMA7 Reset.req primitive. After obtaining all associated confirmations, FMA7 transfers a FMA7 Reset.con(+/-) to the user. FMS, LLI, FMA1/2, FDL and PHY conduct a restart after the reset.

Table 21. FMA7 Reset

Parameter Name	.req	.con
Argument	M	
Result(+)		S
Result(-)		S
Error Type		M

Argument
The argument contains no parameters.

Result(+)
The Result(+) parameter indicates that the service was executed successfully.

Result(-)
The Result(-) parameter indicates that the service failed.

Error Type
This parameter contains information on the reason the service failed.

4.6.2 Network Events

4.6.2.1 Model Description

A service which indicates events and faults in LLI, FDL and PHY is provided to the FMA7 user.

4.6.2.2 Services

The FMA7 Event service is mandatory.

4.6.2.2.1 FMA7 Event

Events and faults which occurred in the LLI, FDL or PHY are notified to the user with the FMA7 event service. The FMA7 does not perform fault handling.

Table 22. FMA7 Event

Parameter Name	.ind
Argument	
Communication Reference	M
Instance Identifier	M
Event/Fault	M
Additional Detail	C

Argument
The argument contains the parameters of the FMA7 Event.ind primitive.

Communication Reference
This parameter specifies the identifier of the associated communication relationship. If the FMA7 Event.ind cannot be assigned to a communication reference, then the value NIL is given.

Instance Identifier
This parameter indicates in which instance the fault or the event occurred.

Valid values
- LLI
- FDL
- PHY

Event/Fault
This parameter specifies the event or the fault. If the parameter Instance Identifier has the value LLI, then the meaning of the parameter Event/Fault shall be found in Table 111 Reason Codes for LLI Fault.ind.
If the parameter Instance Identifier has the value FDL or PHY, then the meaning of the Event/Fault shall be found in part 3.

Additional Detail
This parameter contains additional information on the corresponding event or fault. If the parameter Instance Identifier has the value LLI then the meaning of the parameter Additional Detail shall be found in Table 111 Reason Codes for LLI Fault.ind (tag: AD).
If the parameter Instance Identifier has the value FDL or PHY, then the meaning of the parameter Additional Detail shall be found in the Data Link Layer (Event FMA1/2) of the PROFIBUS Specification.

4.7 Assignment of Services to Master/Slave and Services to Objects

The following table shows which services shall be performed by a passive station (slave) and by an active station (master) and which services are optional.

Table 23. Assignment of Services to Master and Slave

Service	Master				Slave			
	.req	.con	.ind	.res	.req	.con	.ind	.res
FMA7 Initiate	O	O	O	O	-	-	O	O
FMA7 Abort	O	-	O	-	O	-	O	-
InitiateLoad-CRL-Rem	O	O	O	O	-	-	O	O
Load-CRL-Rem	O	O	O	O	-	-	O	O
TerminateLoad-CRL-Rem	O	O	O	O	-	-	O	O
Read-CRL-Rem	O	O	O	O	-	-	O	O
SetValueRem	O	O	O	O	-	-	O	O
ReadValueRem	O	O	O	O	-	-	O	O
LSAP-StatusRem	O	O	O	O	-	-	O	O
IdentRem	O	O	O	O	-	-	O	O
InitiateLoad-CRL-Loc	O	O	-	-	O	O	-	-
Load-CRL-Loc	O	O	-	-	O	O	-	-
TerminateLoad-CRL-Loc	O	O	-	-	O	O	-	-
Read-CRL-Loc	O	O	-	-	O	O	-	-
SetValueLoc	O	O	-	-	O	O	-	-
ReadValueLoc	O	O	-	-	O	O	-	-
LSAP-StatusLoc	O	O	-	-	O	O	-	-
IdentLoc	O	O	-	-	O	O	-	-
GetLiveList	O	O	-	-	O	O	-	-
FMA7 Reset	M	M	-	-	M	M	-	-
FMA7 Event	-	-	M	-	-	-	M	-

Explanation:
M : Mandatory service
O : Optional service
- : not allowed

The following table shows on which objects each service operates.

Table 24. Assignment Services/Objects

Services	Objects														
FMA7 Initiate															
FMA7 Abort															
InitiateLoad-CRL-Rem															
Load-CRL-Rem															
TerminateLoad-CRL-Rem															
Read-CRL-Rem															
SetValueRem															
ReadValueRem															
LSAP-StatusRem															
IdentRem															
InitiateLoad-CRL-Loc	FC	FH	FE	C		H		E							
Load-CRL-Loc	FC	FH	FE	C		H		E							
TerminateLoad-CRL-Loc	FC	FH	FE	C		H		E							
Read-CRL-Loc	FC	FH	FE	C		H		E							
SetValueLoc														P	
ReadValueLoc														P	
LSAP-StatusLoc									L						
IdentLoc										IL	S				
GetLiveList													LL		
FMA7 Reset															
FMA7 Event															

Explanation:
FC : FMA7 CRL
FH : FMA7 CRL Header
FE : FMA7 CRL Entry
C : CRL
H : CRL Header
E : CRL Entry
L : LSAP
IL : Ident List
S : Station Ident List
LL : Live List
P : PHY/FDL Variable

Mapping of FMA7 Services on FMS, LLI, FDL and FMA1/2

The remote FMA7 services are mapped onto the LLI services ASS, DTC and ABT. FMA7 shall set the parameter LLI SAP to the value 1 in the request of these services at the FMA7-LLI interface.

Mapping of FMA7 Context Management Services

FMA7 Initiate:

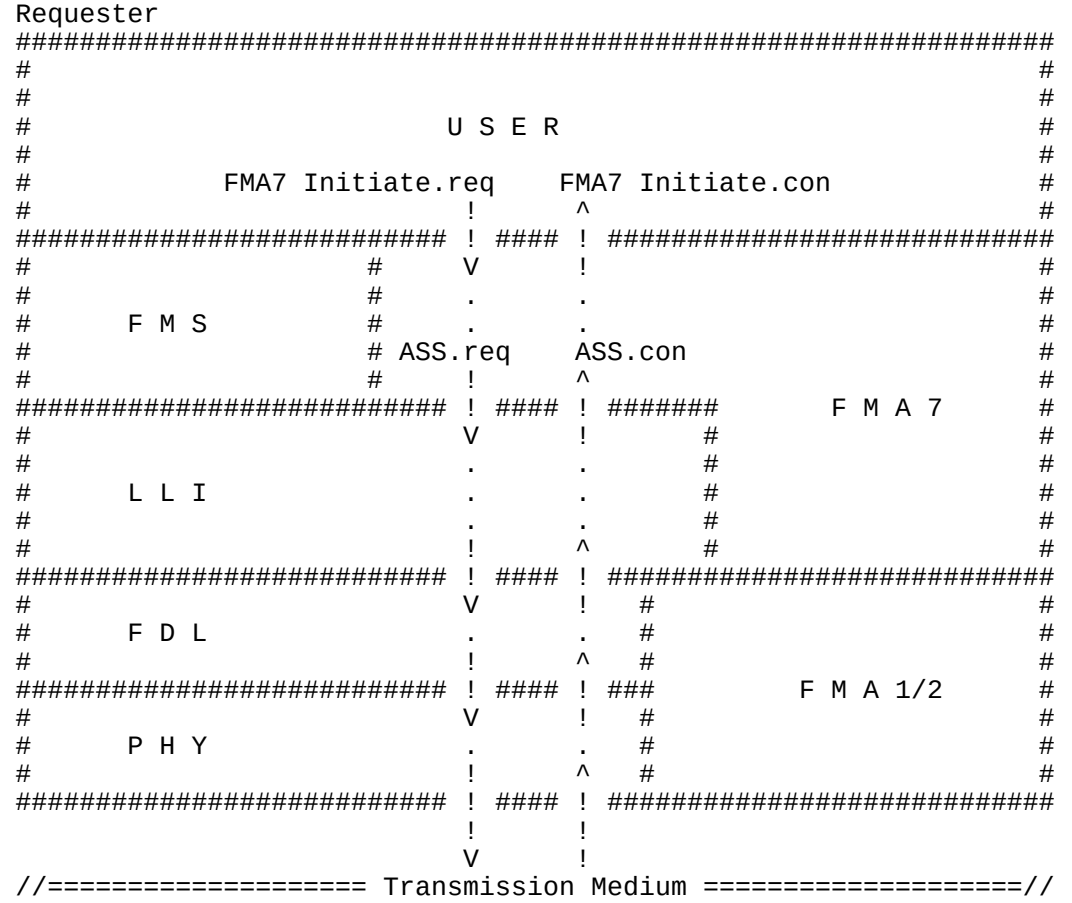


Figure 27. Mapping of the FMA7 Initiate Service at the Requester

FMA7 Initiate:

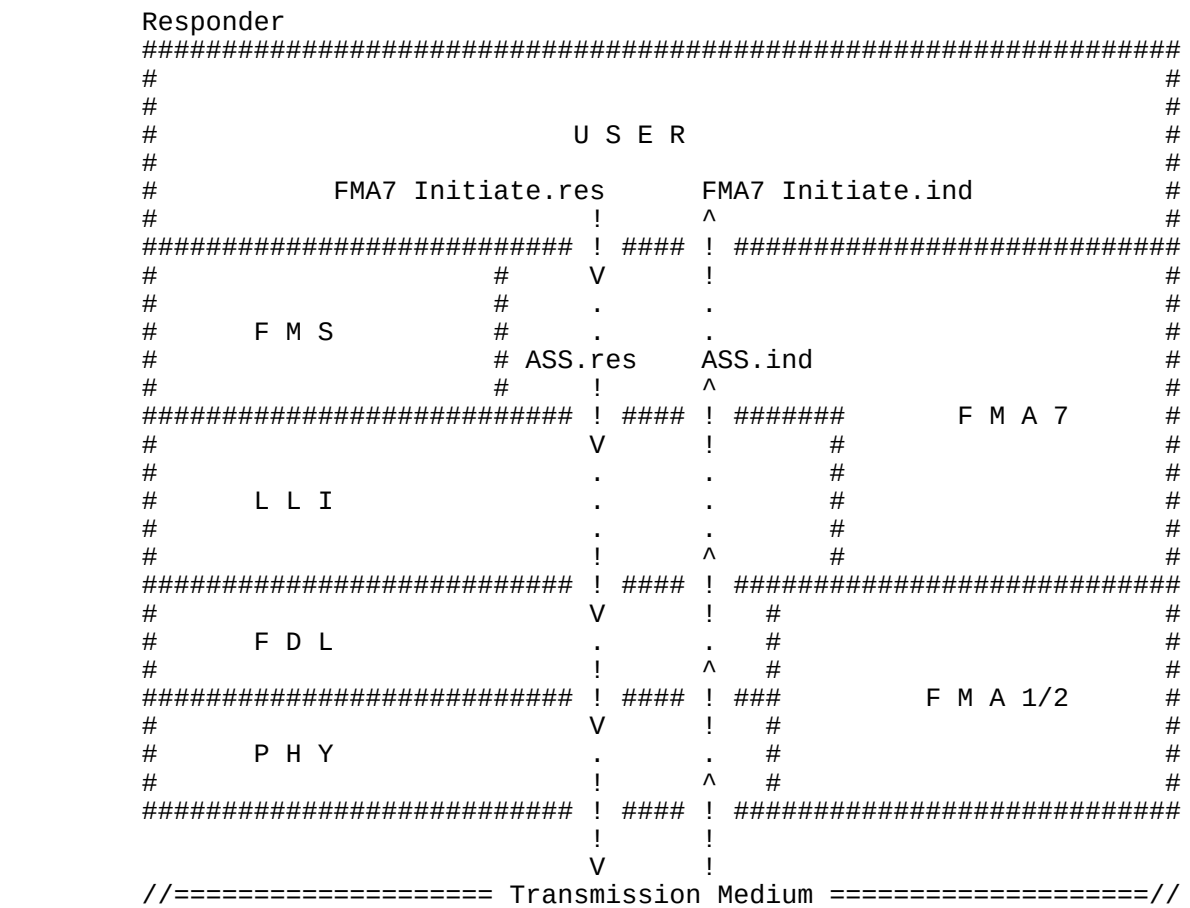


Figure 28. Mapping of the FMA7 Initiate Service at the Responder

FMA7 Abort:

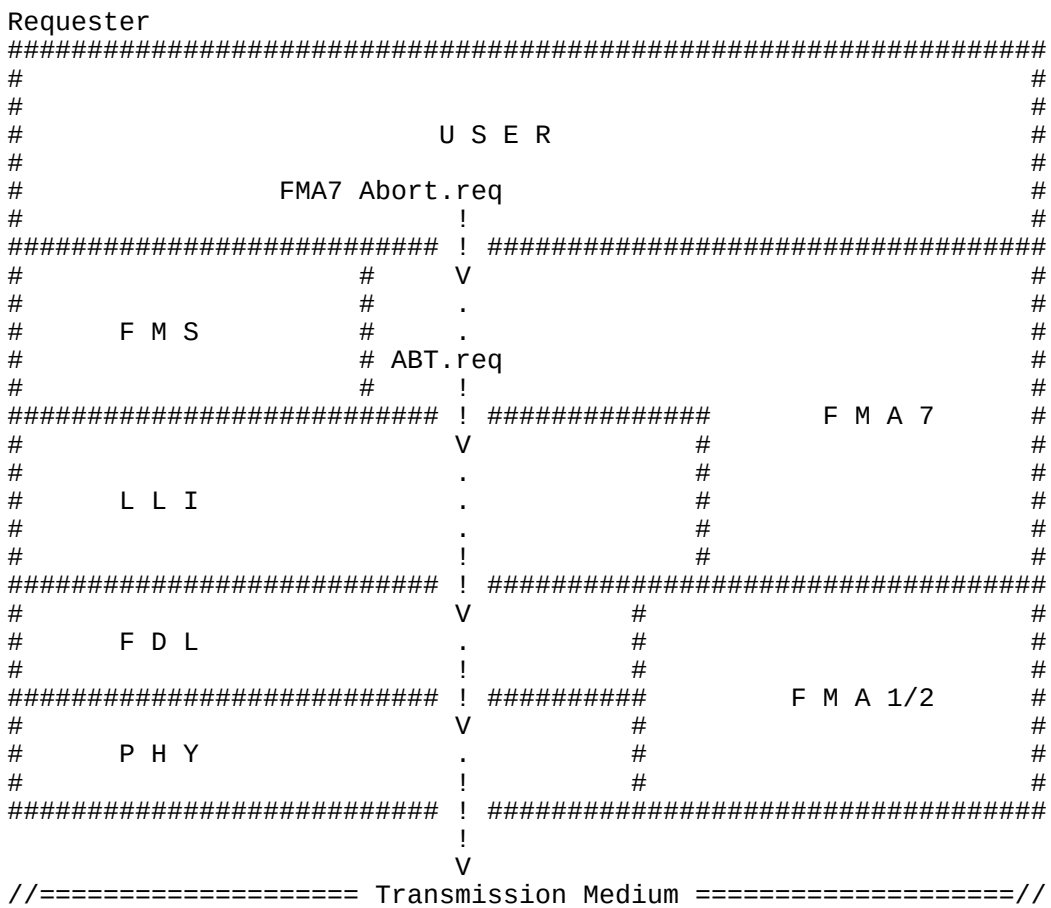


Figure 29. Mapping of the FMA7 Abort Service at the Requester

FMA7 Abort:

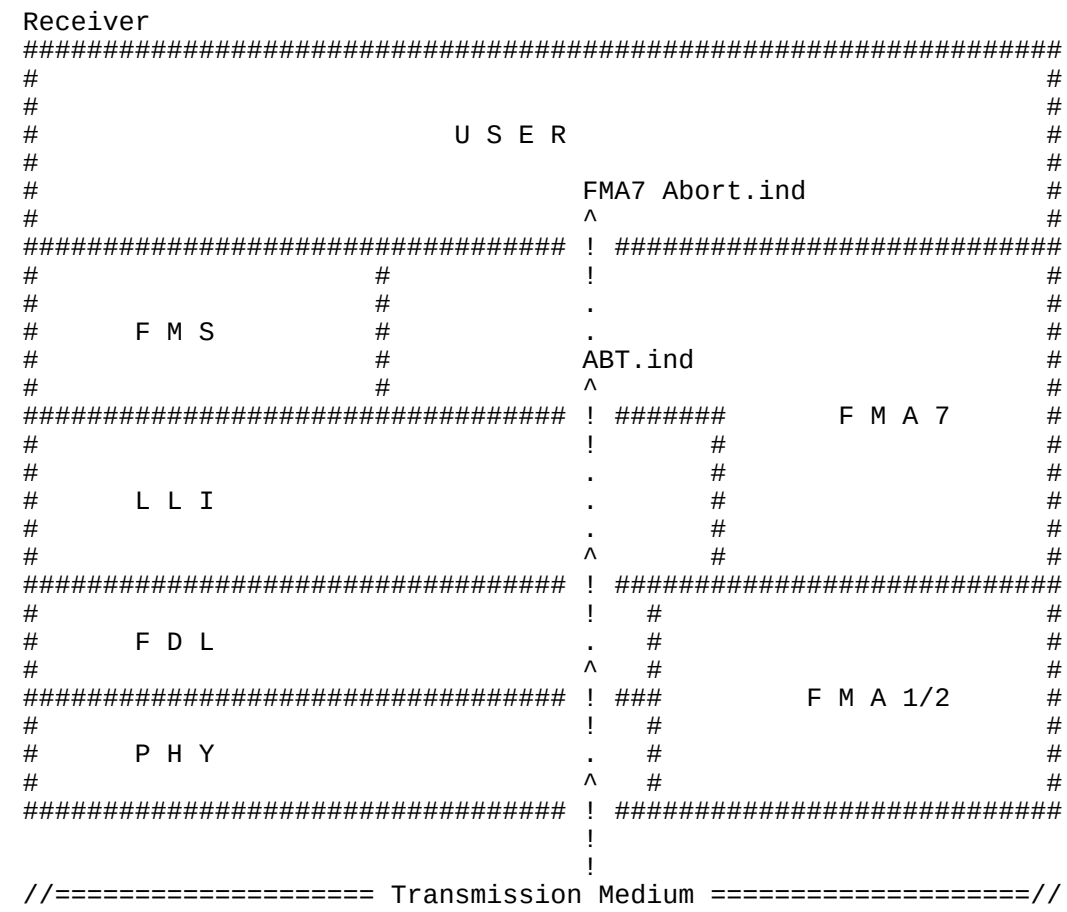


Figure 30. Mapping of the FMA7 Abort Service at the Receiver

4.7.1 Mapping of FMA7 Local Management Services

InitiateLoad-CRL-Loc, Load-CRL-Loc, TerminateLoad-CRL-Loc,

Read-CRL-Loc:

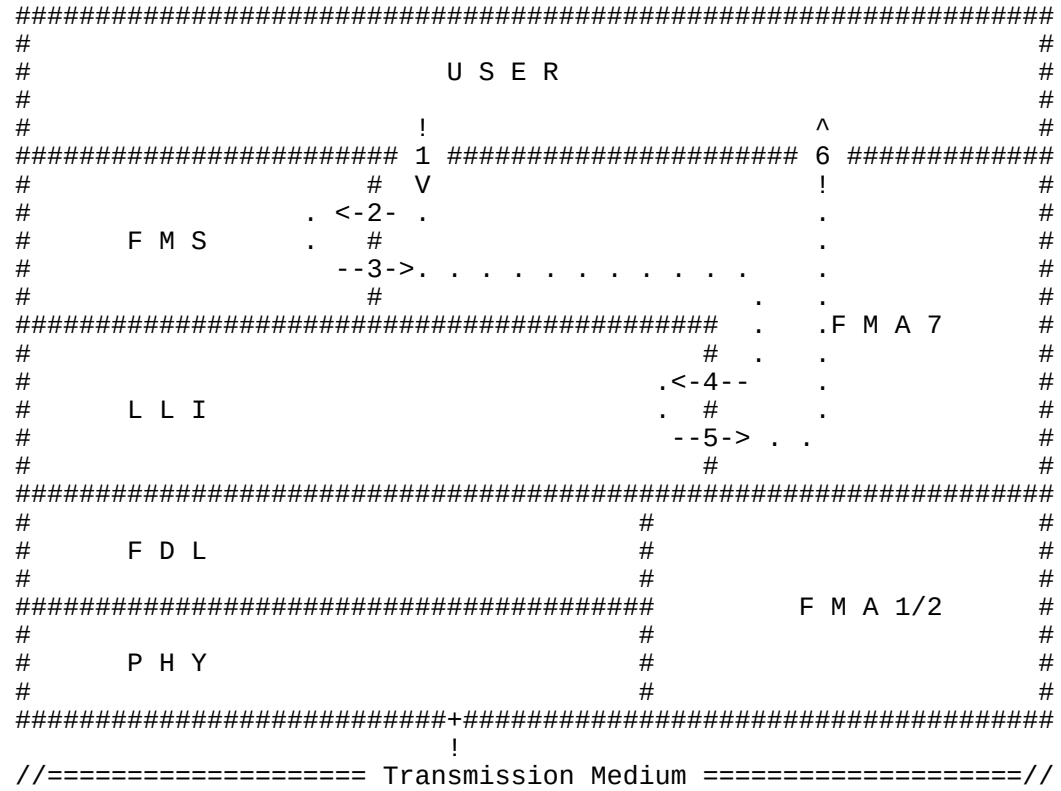


Figure 31. Mapping of the local CRL Management Services

InitiateLoad-CRL-Loc:

1	InitiateLoad-CRL-Loc.req	4	LLI Disable.req
2	FMS Disable.req	5	LLI Disable.con
3	FMS Disable.con	6	InitiateLoad-CRL-Loc.con

Load-CRL-Loc:

1	Load-CRL-Loc.req	4	LLI Load-CRL.req
2	FMS Load-CRL.req	5	LLI Load-CRL.con
3	FMS Load-CRL.con	6	Load-CRL-Loc.con

TerminateLoad-CRL-Loc:

1	TerminateLoad-CRL-Loc.req	4	LLI Enable.req
2	FMS Enable.req	5	LLI Enable.con
3	FMS Enable.con	6	TerminateLoad-CRL-Loc.con

Read-CRL-Loc:

1	Read-CRL-Loc.req	4	LLI Read-CRL.req
2	FMS Read-CRL.req	5	LLI Read-CRL.con
3	FMS Read-CRL.con	6	Read-CRL-Loc.con

SetValueLoc, ReadValueLoc, LSAP-StatusLoc, GetLiveList:

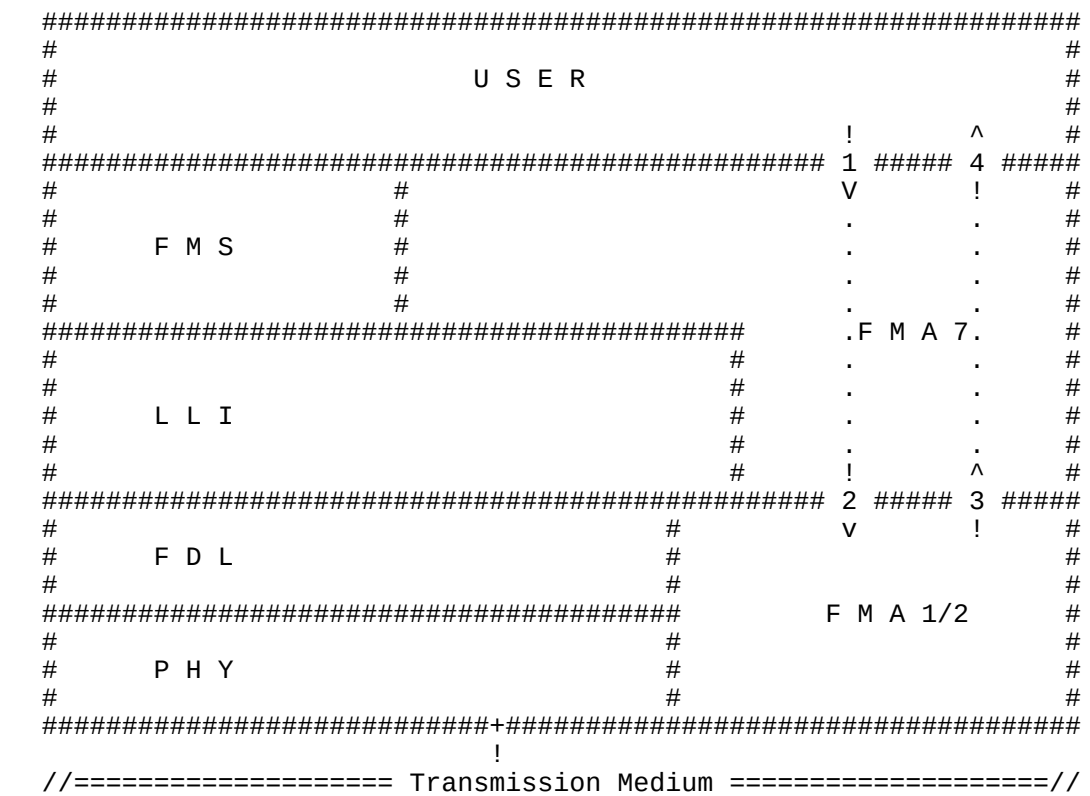


Figure 32. Mapping of the local Services SetValueLoc, ReadValueLoc, LSAP-StatusLoc and GetLiveList

SetValueLoc:
1 SetValueLoc.req
2 FMA1/2 SetValue.req
3 FMA1/2 SetValue.con
4 SetValueLoc.con

ReadValueLoc:
1 ReadValueLoc.req
2 FMA1/2 ReadValue.req
3 FMA1/2 ReadValue.con
4 ReadValueLoc.con

LSAP-StatusLoc:
1 LSAP-StatusLoc.req
2 FMA1/2 LSAP-Status.req(Rem/Loc-add = TS)
3 FMA1/2 LSAP-Status.con
4 LSAP-StatusLoc.con

GetLiveList:
1 GetLiveList.req
2 FMA1/2 LiveList.req
3 FMA1/2 LiveList.con
4 GetLiveList.con

IdentLoc (Instance Identifier = FDL):

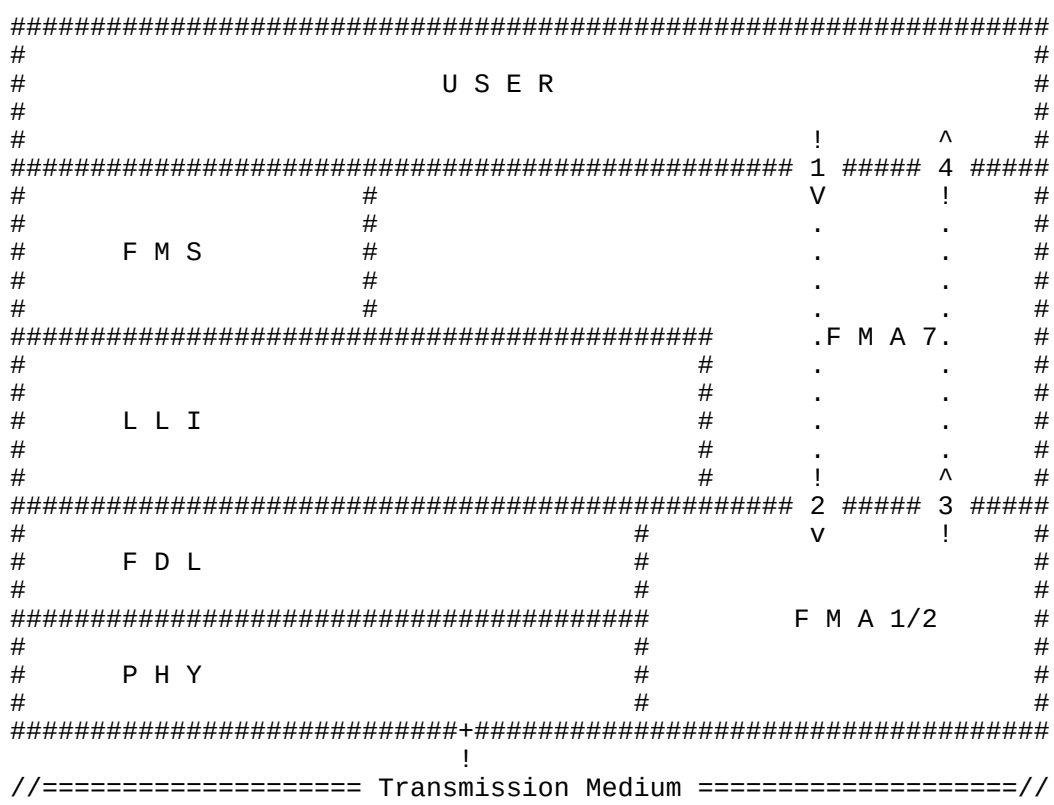


Figure 33. Mapping of the IdentLoc Service (Instance Identifier = FDL)

IdentLoc (Instance Identifier = FDL):
1 IdentLoc.req
2 FMA1/2 Ident.req(Rem/Loc-add = TS)
3 FMA1/2 Ident.con
4 IdentLoc.con

IdentLoc (Instance Identifier = LLI):

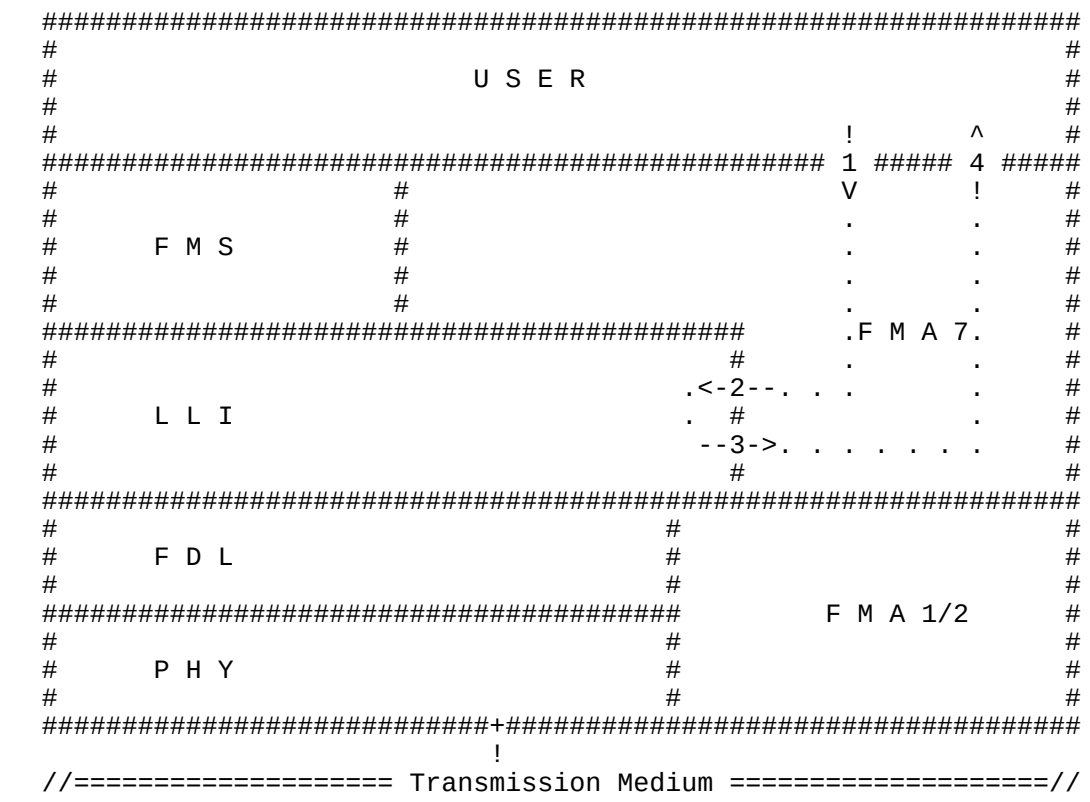


Figure 34. Mapping of the IdentLoc Service

(Instance Identifier = LLI)

IdentLoc (Instance Identifier = LLI):
1 IdentLoc.req
2 LLI Ident.req
3 LLI Ident.con
4 IdentLoc.con

IdentLoc (Instance Identifier = FMS):

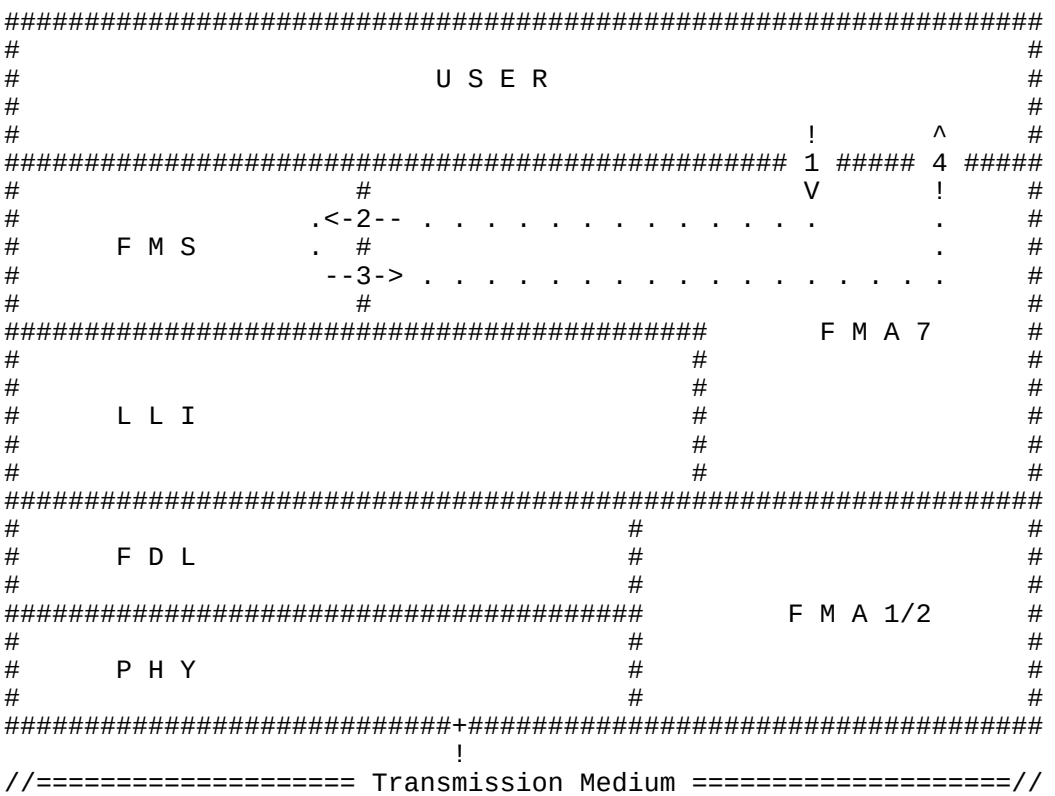


Figure 35. Mapping of the IdentLoc Service (Instance Identifier = FMS)

IdentLoc (Instance Identifier = FMS):
1 IdentLoc.req
2 FMS Ident.req
3 FMS Ident.con
4 IdentLoc.con

IdentLoc (Instance Identifier = FMA7):

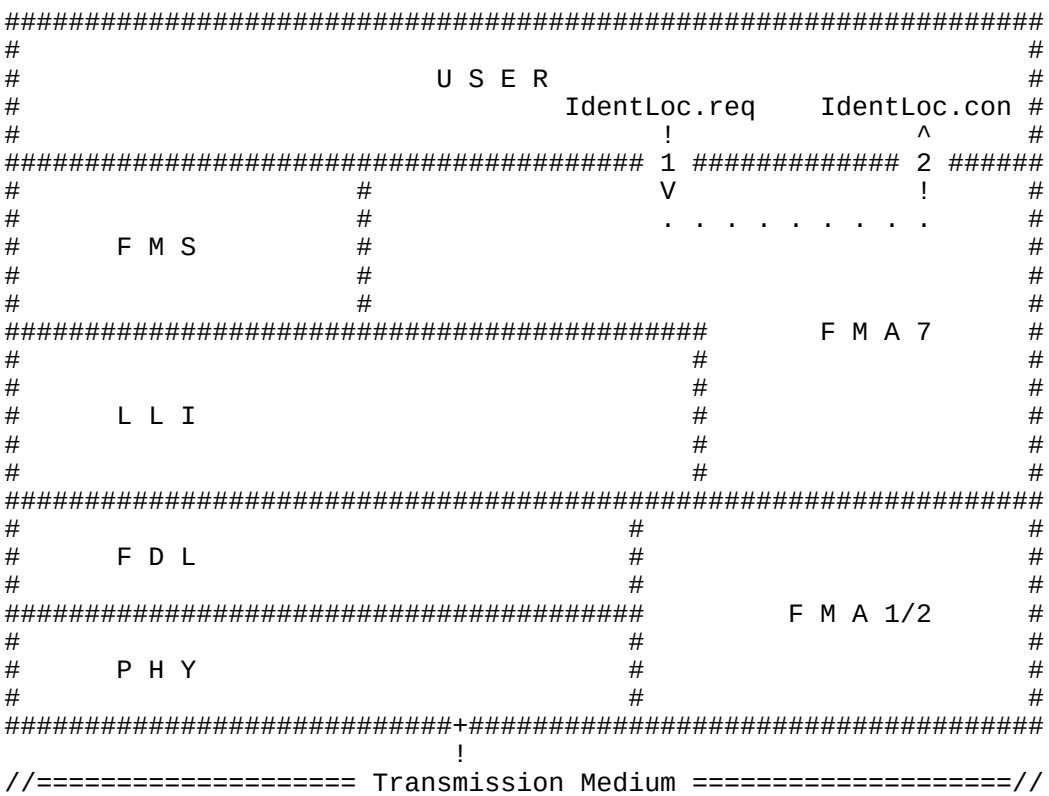


Figure 36. Mapping of the IdentLoc Service (Instance Identifier = FMA7)

FMA7 Reset:

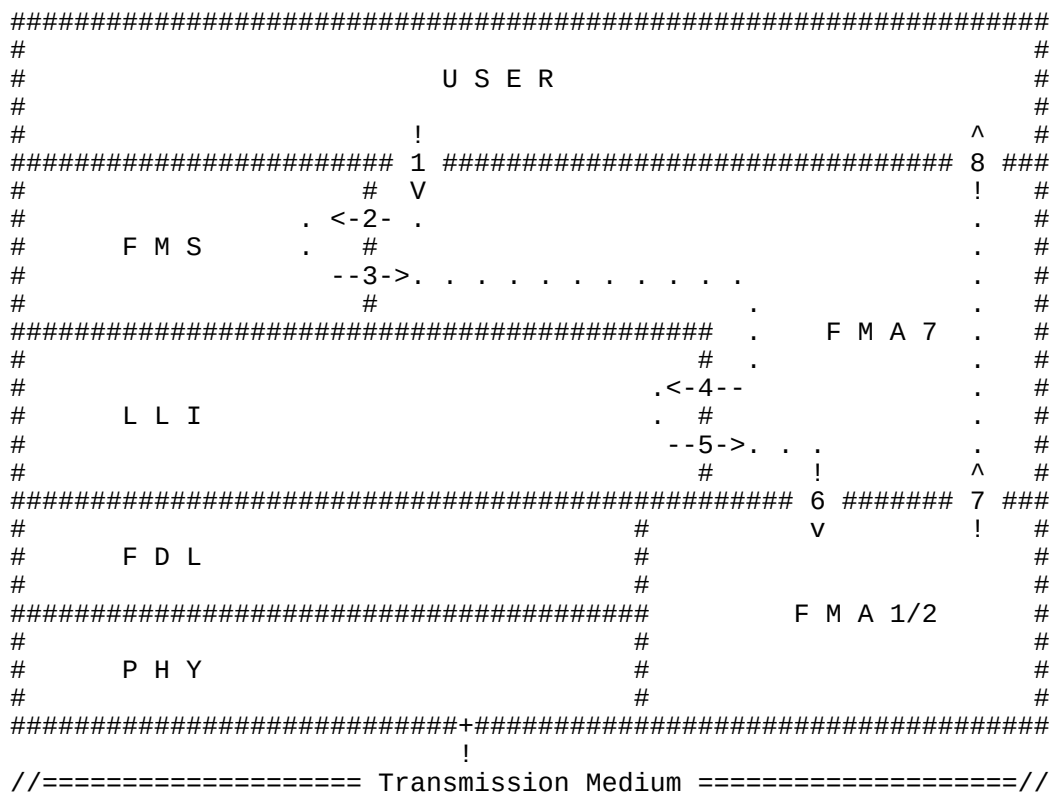


Figure 37. Mapping of the FMA7 Reset Service

FMA7 Reset:
1 FMA7 Reset.req
2 FMS Reset.req
3 FMS Reset.con
4 LLI Reset.req
5 LLI Reset.con
6 FMA1/2 Reset.req
7 FMA1/2 Reset.con
8 FMA7 Reset.con

FMA7 Event (Instance Identifier = PHY/FDL)

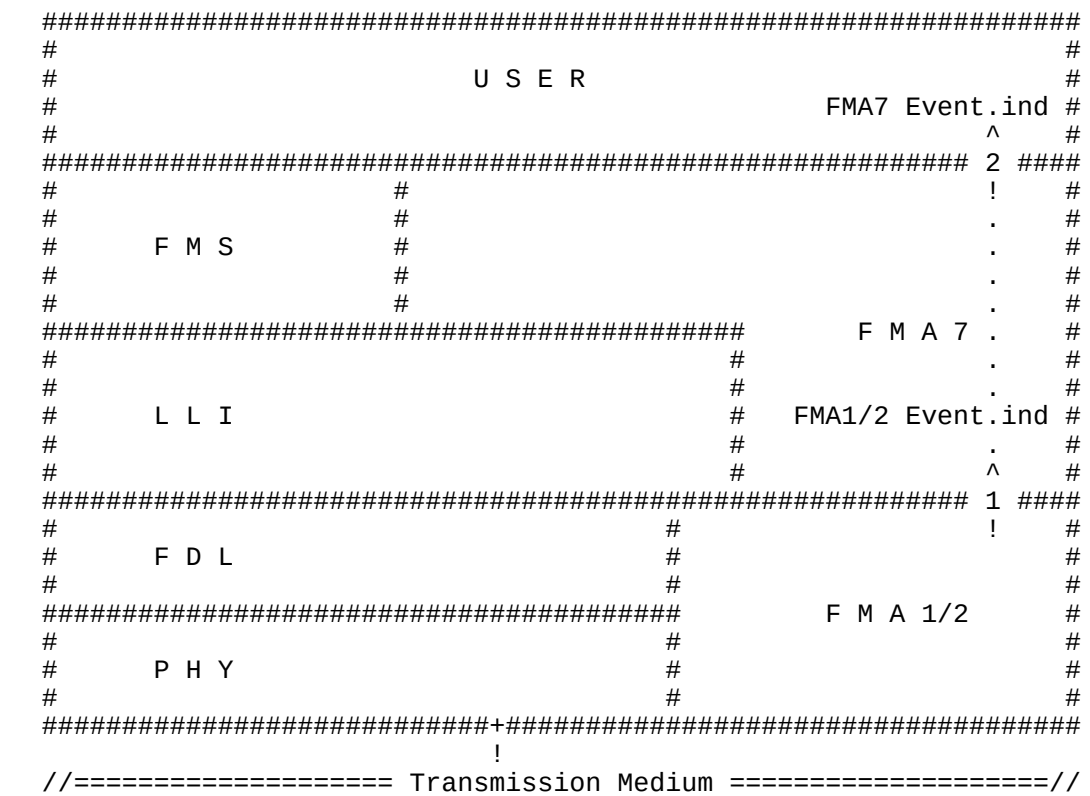


Figure 38. Mapping of the FMA7 Event Service (Instance Identifier = PHY/FDL)

FMA7 Event (Instance Identifier = LLI)

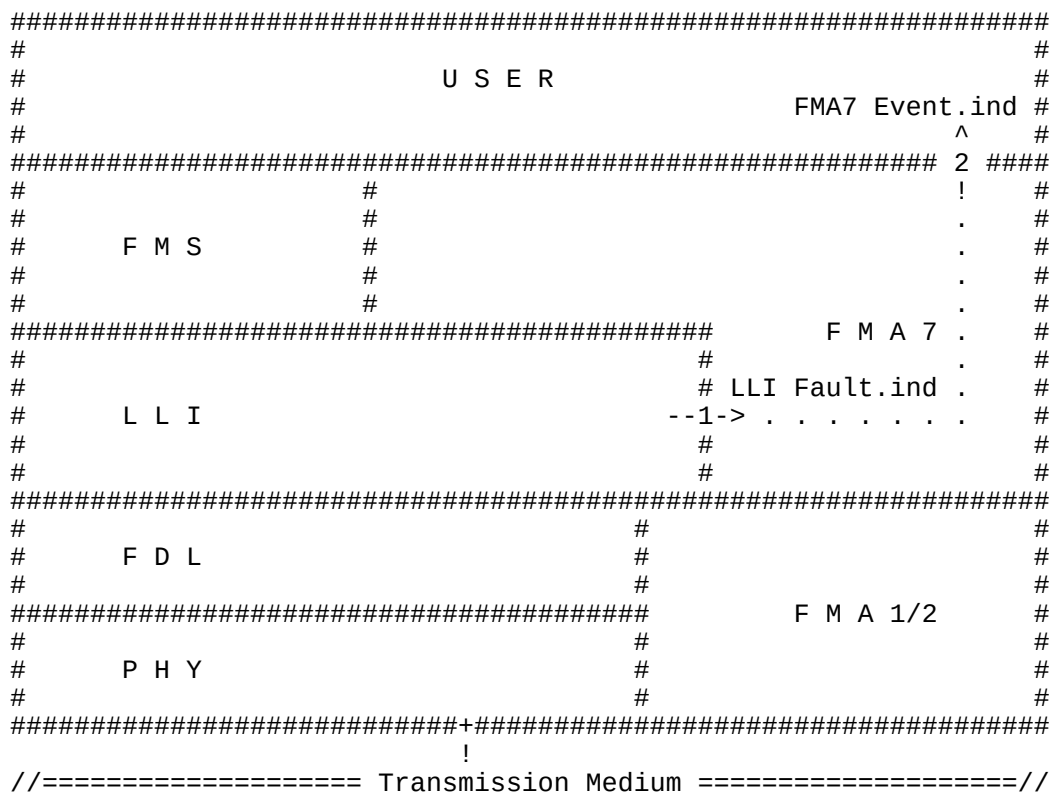


Figure 39. Mapping of the FMA7 Event Service (Instance Identifier = LLI)

4.7.2 Mapping of FMA7 Remote Management Services

FMA7 ServiceRem:

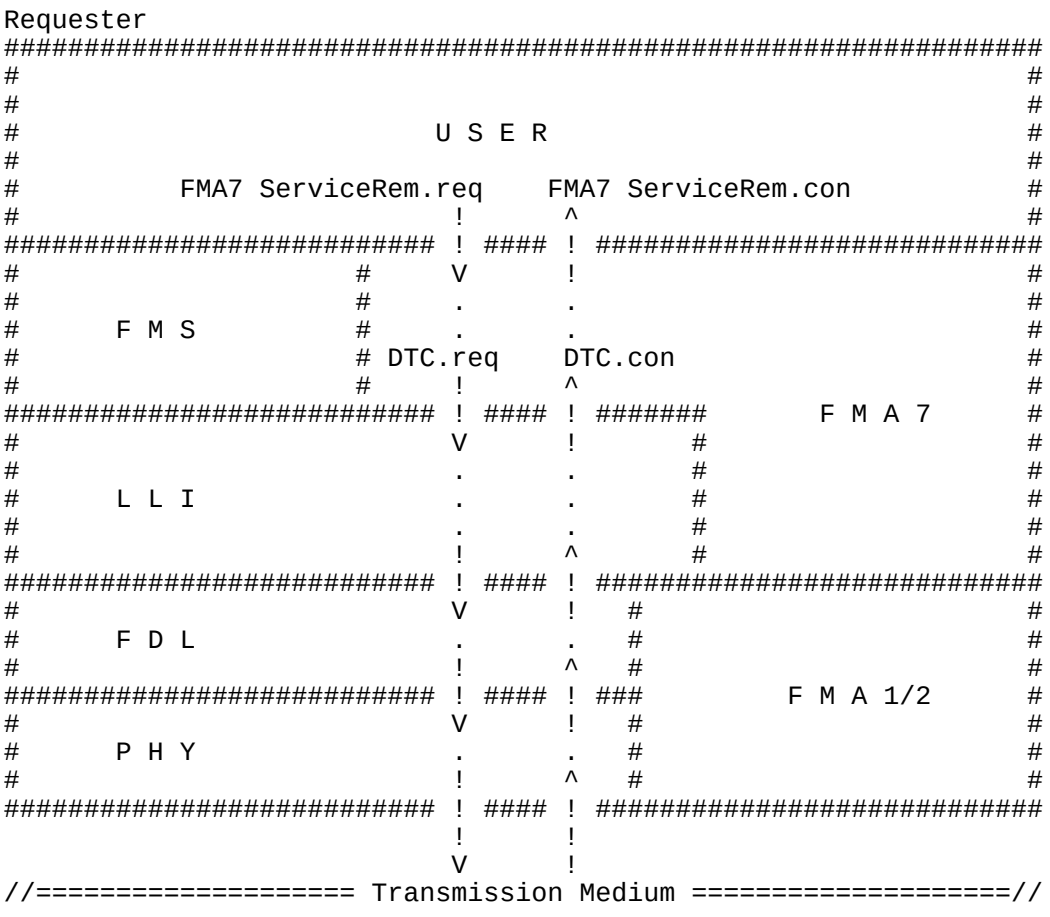


Figure 40. Mapping of the FMA7 Remote Management Services at the Requester

All remote FMA7 services are mapped in the same way at the requester.

FMA7 ServiceRem:

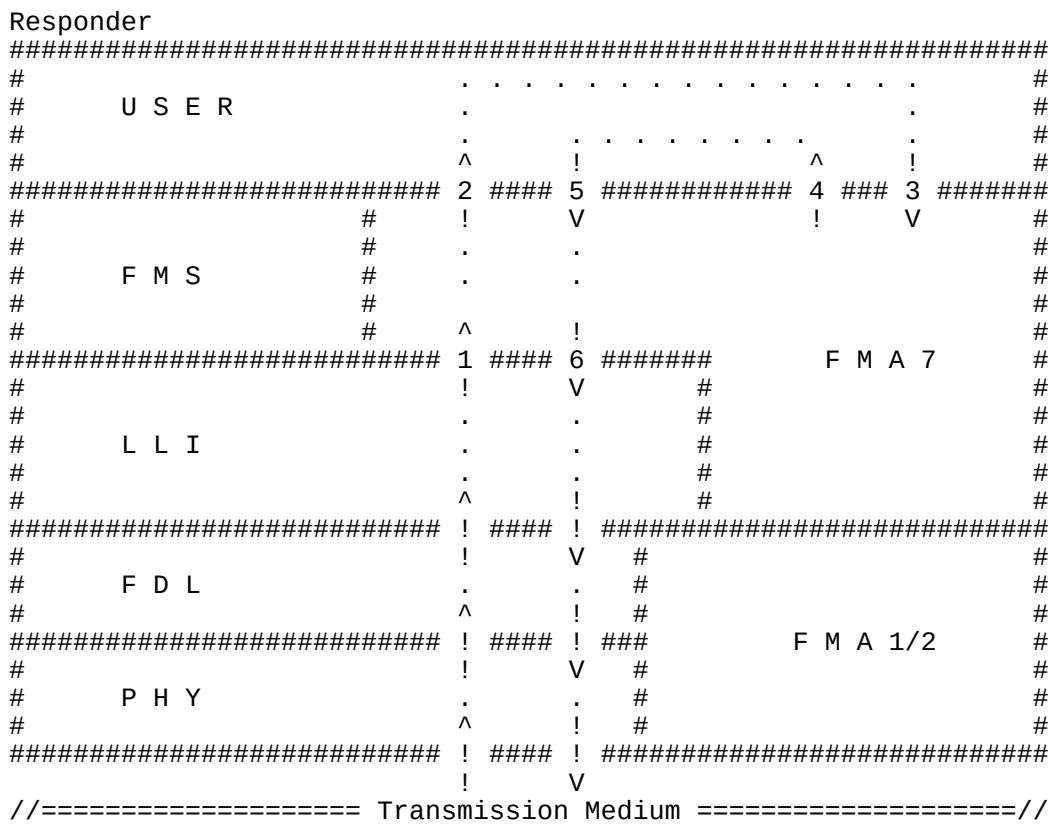


Figure 41. Mapping of the FMA7 Remote Management Services at the Responder

1 DTC.ind
2 FMA7 ServiceRem.ind
3 FMA7 ServiceLoc.req
4 FMA7 ServiceLoc.con
5 FMA7 ServiceRem.res
6 DTC.res

All remote FMA7 services are achieved by the corresponding local services at the responder.

4.8 List of Object Attributes and Service Parameters

Table 25. Object Attributes and Service Parameters

Attribute/Parameter	Data Type
ASS/ABT CI	Unsigned32
Abort Detail	Octet String, max. 16 octets
Abort Identifier	Unsigned8
Access	Unsigned8
Access rights supported	Bit String
Actual Remote Address	Unsigned8
Actual Remote LSAP	Unsigned8
Additional Code	Integer8
Additional Description	Visible String
Additional Detail	--
Number of FMA7 CRL Entries	--
Number of CRL Entries	Integer16
Baud_rate	Unsigned8
CI	Unsigned32
Characteristics	Packed, 54 octets
Controller Type	Visible String
Data types supported	Bit String
DTA-Ack.	Unsigned8
DTA-Req.	Unsigned8
DTC-Req.	Unsigned8
DTC-Res.	Unsigned8
DTU-Rcv.	Unsigned8
DTU-Req.	Unsigned8
FMA7 Error Class	Integer8
FMA7 Error Code	Integer8
Error CREF	Integer16
Error Type	--
Event/Fault	--
FDL Address	Unsigned8
FMA7 CRL Header	--
FMA7 Services Supported	Bit String, 6 octets
Frame_sent_count	Unsigned32
Functions Supported	Bit String, 3 octets
G	Unsigned8
GAPL	Packed
HSA	Unsigned8
HW-Release	Visible String
Hardware Release	Visible String
IDLE.req	Unsigned8
In_ring_desired	Boolean
Instance Identifier	--
CREL State	--
CRL Entry	Octet String
CRL Entry Static	Octet String
CRL Loader Status	--
CRL Upload Status	--
Communication Reference	Unsigned16
LAS	Packed
LLI SAP	Unsigned8
LSAP Value	Unsigned8
LSAP_Status_Data_Unit	Octet String
List of FMA7 CRL Entry	--
Live List Entry	Octet String
Local LSAP	Unsigned8
Locally Generated	Boolean
Loop	Boolean

Table 25. Continuation

Attribute/Parameter	Data Type
max RAC	Unsigned8
max RCC	Unsigned8
max SAC	Unsigned8
max SCC	Unsigned8
MAX_TSDR	Unsigned16
MIN_TSDR	Unsigned16
MAX-TSDR-ABS	Unsigned8
MIN-TSDR-ABS	Unsigned8
Max FMA7 PDU Receiving High Prio	Unsigned8
Max FMA7 PDU Receiving Low Prio	Unsigned8
Max FMA7 PDU Sending High Prio	Unsigned8
Max FMA7 PDU Sending Low Prio	Unsigned8
Max Outstanding Services Client	Unsigned8
Max Outstanding Services Server	Unsigned8
Max PDU Receiving High Prio	Unsigned8
Max PDU Receiving Low Prio	Unsigned8
Max PDU Sending High Prio	Unsigned8
Max PDU Sending Low Prio	Unsigned8
Medium_red	Unsigned8
Multiplier	Unsigned8
Outstanding FMA7 Services Counter Req	--
Outstanding FMA7 Services Counter Res	--
Poll Entry Enabled	Boolean
Poll List SSAP	Unsigned8
RAC	Unsigned8
RCC	Unsigned8
Reason Code	Unsigned8
Received_signal_source	Unsigned8
Remote Address	Unsigned8
Remote LSAP	Unsigned8
Retry_count	Unsigned16
SAC	Unsigned8
SCC	Unsigned8
SD_count	Unsigned32
SD_error_count	Unsigned16
SW-Release	Visible String
Service_activate	Octet String
Services Supported	Bit String, 6 octets
Software Release	Visible String
Special Functions Supported	Bit Strings
Desired CREF	Integer16
Status	Octet String
Symbol	Visible String, max. 32 octets
Symbol Length	Unsigned8
Tout-ABS	Unsigned8
TQUI	Unsigned8
TRDY-ABS	Unsigned8
TS	Unsigned8
TSDI-ABS	Unsigned8
TSET	Unsigned8
Tset-ABS	Unsigned8
TSL	Unsigned16
TTR	Unsigned32
Station Type	Unsigned8
Transmitter_output	Boolean
Type	Unsigned8
CN Estab. / CN Release / Open	Unsigned8
Variable types supported	Bit Strings
VFD Pointer	Unsigned32
VFD Pointer Supported	Boolean

4.9 Syntax Description

The encoding of the FMA7 PDUs shall be done according to the encoding rules for FMS (see FMS coding).

4.9.1 The FMA7 PDU

Like the FMS PDU, the FMA7 PDU consists of a fixed part of 3 octets and a part of variable length. Not all PDUs require a part of variable length.

The fixed part consists of a First ID Info which specifies the service class (e.g. confirmed request, confirmed response), of an Invoke ID (1 octet, data type Integer8) and of a Second ID Info which identifies the PDU more precisely.

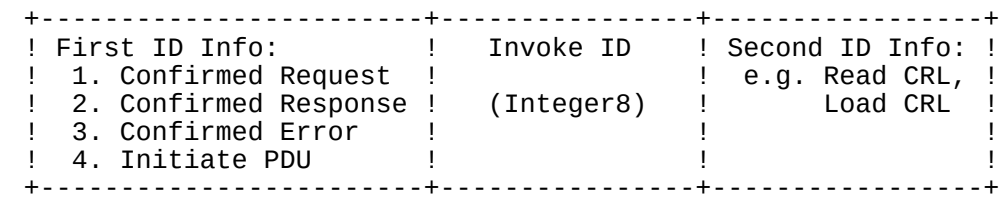


Figure 42. Field with fixed Format

The fixed format of the first 3 octets defines a location for the Invoke ID of the FMA7 PDUs, which is not occupied (as for the FMS PDUs Unconfirmed, Reject and Initiate).

In the following the fixed part of the FMA7 PDU is described:

```
FMA7pdu ::= CHOICE {
  confirmed-FMA7-request-PDU   [1] IMPLICIT Confirmed-FMA7-Request-PDU,
  confirmed-FMA7-response-PDU  [2] IMPLICIT Confirmed-FMA7-Response-PDU,
  confirmed-FMA7-error-PDU     [3] IMPLICIT Confirmed-FMA7-Error-PDU,
  FMA7-initiate-PDU            [4] IMPLICIT FMA7-Initiate-PDU
  }

  Confirmed-FMA7-Request-PDU ::= SEQUENCE {
    invokeID                        Invoke-ID, -- Dummy
    confirmed-FMA7-service-request   Confirmed-FMA7-Service-Request

    }

  Confirmed-FMA7-Response-PDU ::= SEQUENCE {
    invokeID                        Invoke-ID, -- Dummy
    confirmed-FMA7-service-response  Confirmed-FMA7-Service-Response


    }
  Confirmed-FMA7-Error-PDU ::= SEQUENCE {
    invokeID                        Invoke-ID, -- Dummy
    FMA7-service-error              FMA7-Service-Error
    }

FMA7-Initiate-PDU ::= SEQUENCE {
    invokeID            Invoke-ID,  -- Dummy
    FMA7-initiate       FMA7-Initiate
    }
```

4.9.2 Confirmed Service Request and Response

```
Confirmed-FMA7-Service-Request ::= CHOICE {
  read-crl-rem                  [ 0] IMPLICIT Read-CRL-Rem- Request,
  initiate-load-crl-rem         [ 1] IMPLICIT
                                       Initiate-Load-CRL-Rem-Request,
  load-crl-rem                  [ 2] IMPLICIT Load-CRL-Rem-Request,
  terminate-load-crl-rem        [ 3] IMPLICIT
                                       Terminate-Load-CRL-Rem-Request,
  set-value-rem                 [ 4] IMPLICIT Set-Value-Rem-Request,
  read-value-rem                [ 5] IMPLICIT Read-Value-Rem- Request,
  lsap-status-rem               [ 6] IMPLICIT LSAP-Status-Rem-Request,
  ident-rem                     [ 7] IMPLICIT Ident-Rem-Request
  }


Confirmed-FMA7-Service-Response ::= CHOICE {
  read-crl-rem                  [ 0] IMPLICIT  Read-CRL-Rem-Response,
  initiate-load-crl-rem         [ 1] IMPLICIT
                                      Initiate-Load-CRL-Rem-Response,
  load-crl-rem                  [ 2] IMPLICIT Load-CRL-Rem-Response,
  terminate-load-crl-rem        [ 3] IMPLICIT
                                      Terminate-Load-CRL-Rem-Response,
  set-value-rem                 [ 4] IMPLICIT Set-Value-Rem-Response,
  read-value-rem                [ 5] IMPLICIT Read-Value-Rem-Response,
  lsap-status-rem               [ 6] IMPLICIT LSAP-Status-Rem-Response,
  ident-rem                     [ 7] IMPLICIT Ident-Rem-Response
  }
```

4.9.2.1 Read-CRL-Rem

```
Read-CRL-Rem-Request ::= Desired-CREF
Desired-CREF ::= Unsigned16
Read-CRL-Rem-Response ::= CRL-Eintrag
```

4.9.2.2 InitiateLoadCRL-Rem

```
Initiate-Load-CRL-Rem-Request ::= NULL
Initiate-Load-CRL-Rem-Response ::= NULL
```

4.9.2.3 Load-CRL-Rem

```
Load-CRL-Rem-Request ::= CRL-Eintrag
Load-CRL-Rem-Response ::= NULL
```

4.9.2.4 TerminateLoad-CRL-Rem

```
Terminate-Load-CRL-Rem-Request ::= NULL
Terminate-Load-CRL-Rem-Response ::= NULL
```

4.9.2.5 SetValueRem

```
Set-Value-Rem-Request ::= Sequence {
              variable-identifier     [0] IMPLICIT Unsigned8,
                                           -- encoding see
                                           -- subclause PHY/FDL Variables
              desired-value           [1] IMPLICIT OCTET STRING
              }
Set-Value-Rem-Response ::= NULL
```

4.9.2.6 ReadValueRem

```
Read-Value-Rem-Request ::= Variable-Identifier
Variable-Identifier ::= Unsigned8  -- encoding see
                                   -- subclause PHY/FDL Variables
Read-Value-Rem-Response ::= Current-Value
Current-Value ::= OCTET STRING -- encoding see
                               -- subclause PHY/FDL Variables
```

4.9.2.7 LSAP-StatusRem

```
LSAP-Status-Rem-Request ::= LSAP
LSAP ::= Unsigned8
LSAP-Status-Rem-Response ::= LSAP-Status-Data-Unit
LSAP-Status-Data-Unit ::= OCTET STRING
```

4.9.2.8 IdentRem

```
Ident-Rem-Request ::= Instance-Identifier
Instance-Identifier ::= Unsigned8 {
    FMA7     (0),
    FMS      (1),
    LLI      (2),
    FDL      (3),
    Station  (4)
                            }
Ident-Rem-Response ::= SEQUENCE {
  vendor-name               [0] IMPLICIT VISIBLE STRING,
  controllertype            [1] IMPLICIT VISIBLE STRING,
  hardware                  [2] IMPLICIT VISIBLE STRING,
  software                  [3] IMPLICIT VISIBLE STRING,
  characteristics           [4] IMPLICIT PACKED
                                  -- encoding see
                                  -- subclause PHY/FDL Variables
  }
```

4.9.3 ServiceError

```
FMA7-Service-Error ::= CHOICE {
  read-crl-rem                  [0] IMPLICIT FMA7-Error,
  initiate-load-crl-rem         [1] IMPLICIT FMA7-Error,
  load-crl-rem                  [2] IMPLICIT FMA7-Error,
  terminate-load-crl-rem        [3] IMPLICIT FMA7-Terminate-Load-
                                      CRL-Error,
  set-value-rem                 [4] IMPLICIT FMA7-Error,
  read-value-rem                [5] IMPLICIT FMA7-Error,
  lsap-status-rem               [6] IMPLICIT FMA7-Error,
  ident-rem                     [7] IMPLICIT FMA7-Error
  }
```

4.9.3.1 FMA7 TerminateLoad-CRL-Error

```
FMA7-Terminate-Load-CRL-Error ::= SEQUENCE {
  fma7-error              [0] FMA7-Error,
  error-cref              [1] CREF OPTIONAL
  }

CREF ::= Unsigned16
```

4.9.3.2 FMA7 Error-Type

```
FMA7-Error ::= SEQUENCE {
  fma7-error-class          [1] FMA7-Error-Class,
  additional-code           [2] Integer8 OPTIONAL,
  additional-description    [3] VISIBLE-STRING OPTIONAL
  }

FMA7-Error-Class ::=  SEQUENCE {
  CHOICE  {
  application-reference    [1] IMPLICIT INTEGER {
      other                              (0),
      application-unreachable            (1)
                                                },
  resource                 [2] IMPLICIT INTEGER {
      other                              (0),
      memory-unavailable                 (1)
                                                },
  service                  [3] IMPLICIT INTEGER {
      other                              (0),
      object-state-conflict              (1),
      object-constraint-conflict         (2),
      parameter-inconsistent             (3),
      illegal-parameter                  (4),
      permanent-internal-fault           (5)
                                                },
  user                     [4] IMPLICIT INTEGER {
      other                              (0),
      don't-worry-be-happy               (1),
      memory-unavailable                 (2)
                                                },
  access                   [5] IMPLICIT INTEGER {
      other                              (0),
      object-access-unsupported          (1),
      object-non-existent                (2),
      object-access-denied               (3),
      hardware-fault                     (4),
      type-conflict                      (5)
                                                },
  crl-error                [6] IMPLICIT INTEGER {
      other                              (0),
      invalid-crl-entry                  (1),
      no-crl-entry                       (2),
      invalid-crl                        (3),
      no-crl                             (4),
      crl-write-protected                (5)
                                                },
  others                   [7] IMPLICIT INTEGER {
      other                              (0)
                                                }
      }
  }
```

4.9.4 FMA7 Initiate

```
FMA7-Initiate ::= CHOICE {
  FMA7-initiate-request     [0] IMPLICIT FMA7-Initiate-Request,
  FMA7-initiate-response    [1] IMPLICIT FMA7-Initiate-Response,
  FMA7-initiate-error       [2] IMPLICIT FMA7-Initiate-Error
  }
```

4.9.4.1 FMA7 Initiate-Request

```
FMA7-Initiate-Request ::= SEQUENCE {
  max-fma7-pdu-sending-low-prio-calling   [0] IMPLICIT Unsigned8,
  max-fma7-pdu-receiving-low-prio-calling [1] IMPLICIT Unsigned8,
  fma7-Services-supported-calling         [2] IMPLICIT BIT STRING
                                             -- encoding see
                                             -- clause CRL Object
  }
```

4.9.4.2 FMA7 Initiate-Response

```
FMA7-Initiate-Response ::= NULL
```

4.9.4.3 FMA7 Initiate-Error

```
FMA7-Initiate-Error ::= SEQUENCE  {
  error-code                              [0] IMPLICIT INTEGER {
    other                       (0),
    max-pdu-size-insufficient   (1),
    service-not-supported       (2),
    user-initiate-denied        (3)
    },
  max-fma7-pdu-sending-low-prio-called   [1] IMPLICIT Unsigned8,
  max-fma7-pdu-receiving-low-prio-called [2] IMPLICIT Unsigned8,
  FMA7-Services-Supported-called         [3] IMPLICIT BIT STRING
  }
```

4.9.5 General Substitutions

```
Invoke-ID ::= Integer8

CRL-Entry ::= PACKED

Integer8  ::= INTEGER  --  8 Bit Integer
Integer16 ::= INTEGER  -- 16 Bit Integer

Unsigned8  ::= UNSIGNED   --  8 Bit  Unsigned
Unsigned16 ::= UNSIGNED   -- 16 Bit  Unsigned
```

4.10 Error Reports

An error report in Result(-) has the following structure:

Table 26. Error Type

Error Type	.res .con
FMA7 Error Class	M
FMA7 Error Code	M
Additional Code	U
Additional Description	U

FMA7 Error Class
This parameter specifies the error class.

FMA7 Error Code
This parameter gives a more precise specification within the error class.

Additional Code (optional)
This parameter is optional. Its use and interpretation is the responsibility of the user.

Additional Description (optional)
This parameter is optional and may be used to add a text to the error report.

4.10.1 Meaning of FMA7 Error Class and FMA7 Error Code

FMA7 Error Class

meaning of the error class

- FMA7 Error Code
  meaning of the error code

Application Reference

This error class refers to the communication relationship on which the service is performed.

- Other
  The error could not be assigned to any of the error codes specified below.
- Application-Unreachable
  The application process is unreachable.

Resource

This error class is reported when the available resources are exceeded.

- Other
  The error could not be assigned to any of the error codes specified below.
- Memory Unavailable
  There is no memory available to perform the service.

Service
This error class is reported in case of a faulty service.

- Other
  The error could not be assigned to any of the error codes specified below.
- Object State Conflict
  The current state of the object prevents an execution of the service.
- Object Constraint Conflict
  The execution of the service is not possible currently.
- Parameter Inconsistent
  The service contains inconsistent parameters.
- Illegal Parameter
  A parameter has an illegal value.
- Permanent Internal Fault
  The service could not be performed because of a permanent internal fault.

User
This error class is reported if the user refuses to execute the service.

- Other
  The error could not be assigned to any of the error codes specified below.
- Don't Worry Be Happy
  This error is depends on the user and shall be defined by profiles.
- Memory Unavailable
  The user has insufficient memory to perform the service.

FMA7 Error Class
meaning of the error class

- FMA7 Error Code
  meaning of the error code

Access
This error class is reported in case of a faulty access.

- Other
  The error could not be assigned to any of the error codes specified below.
- Object Access Unsupported
  The object is not defined for the requested access.
- Object Non Existent
  The object does not exist.
- Object Access Denied
  The FMA7 requester has insufficient access rights for the object.
- Hardware Fault
  The access to the object failed due to a hardware fault.
- Type Conflict
  The access is rejected because of an incorrect data type.

CRL Error

This error class is reported in case of a faulty access to the CRL (e.g. InitiateLoad-CRL-Rem, Load-CRL-Rem, TerminateLoad-CRL-Rem).

- Other
  The error could not be assigned to any of the error codes specified below.
- Invalid CRL Entry
  A CRL entry is invalid.
- No CRL Entry
  No further CRL entry exists.
- Invalid CRL
  The CRL is invalid (e.g. FMS part and LLI part of the CRL are incompatible).
- No CRL
  No CRL exists.
- CRL Write Protected
  The CRL is write protected.

Others

This error class could not be assigned to any of the previously specified error classes.

- Other
  The error could not be assigned to any of the previously specified error codes.

4.10.2 Meaning of remaining Parameters

Additional Code
Specification by profiles

Additional Description
Specification by profiles

This page is intentionally left blank.

PROFIBUS Specification - Normative Parts

Part 8

User Specifications

CONTENTS

1 Field of Application and Purpose

The PROFIBUS Specification comprises a great variety of functions and allows for a wide range of applications. This includes cell communication between powerful controllers as well as interfaces to field devices.
Part 2 to Part 4 define media access and transmission control (Data Link Layer, Layer 2 of the OSI Layer Model, see also DIN 7498) and Physical Layer (Layer 1). This parts of the specification are independent of the application layer functions.
Part 5 and Part 6 define the application layer (Layer 7) functions of this fieldbus specification. This Specification is similar to Manufacturing Message Specification MMS (DIN ISO 9506) and thus makes interfacing to MAP networks (Manufacturing Automation Protocol) feasible.
Part 2 to Part 7 are required in applications that need the high functionality of FMS services and interfacing to MMS.
Applications in the area of decentralized peripherals that do not need the powerful services of FMS but require extremely short system reaction times should use the solution presented in the following paper. The well known properties of Part 2 to Part 4 of this specification are used in combination with a very efficient User-Interface. Thus PROFIBUS-DP expands the sequence of PROFIBUS Standards for this specific application area.

The following definitions describe the necessary functional and electrical properties to manage peripheral devices connected through a serial interface to e. g. a controller in manufacturing applications. This specification describes an abstract model but the real form of an implementation will be not fixed by this specification. Part 2 to Part 4 are used as physical and data link layer and extends its definitions to fulfil the special requirements in the area of remote peripherals.
Decentralized Peripherals (DP) are mainly used to connect automation systems (such as programmable controllers) via a fast serial link to input-/output-devices, sensors, actuators and to smart devices.
The main purpose of PROFIBUS-DP is the fast cyclic exchange of data between a powerful Master (automation system) and several simple Slaves (peripheral devices). Thus, this system uses mainly the Master-Slave-type of communication services.
The hybrid media access of PROFIBUS allows Master-Slave communication as well as Master-Master communication, which is used for Data transfer between DP-Master (class 1) and DP-Master (class 2)(programmer/diagnostic-panel).
In applications that do not need the maximum speed, a combination of PROFIBUS-DP with PROFIBUS-FMS on the same network is possible.

2 Normative References and additional Reference Material

DIN 41 652 Teil 1 (draft)	Steckverbindungen in der Nachrichtentechnik; Steckverbinder für die Einschubtechnik, trapezförmig, runde Kontakte 1 mm
DIN 66003	Informationsverarbeitung; 7-Bit-Code
DIN 66 259 Teil 3	Elektrische Eigenschaften der Schnittstellen-leitungen; Doppelstrom, symmetrisch, bis 10 Mbit/s
DIN EN 61131-2	Programmable Controllers-Part 2: Equipment Requirements and Tests
DIN EN 66306 Teil 1	Industrielle Automation und Integration; Fest-legung der Nachrichtenformate für Fertigungs-zwecke; Definition der Dienste; Identisch mit ISO/IEC 9506-1:1990
DIN ISO 2375	Datenverarbeitung; Verfahren zur Registrierung von "Escape-Sequenzen"
DIN ISO 7498	Informationsverarbeitung, Kommunikation Offener Systeme; Basis-Referenzmodell
EIA RS-485	Standard for electrical Characteristics of Generators and Receivers for use in balanced digital Multipoint Systems 1)

1) Source of Supply for DIN Standards:

Beuth Verlag GmbH
Burggrafenstr. 6

D-10787 Berlin
Germany

3 Abbreviations

AD	Access Denied
ASIC	Application Specific Integrated Circuit
CNTR	Control Signal
DDB	Device Data Base
DGND	Data Ground (PHY, RS-485)
DI	Data Incomplete
DP	Decentralized Peripherals
DS	Disconnected Station local FDL/PHY controller not in logical token ring or disconnected from line
EA	Area too large (Up-/Download)
FDL	Fieldbus Data Link
FE	Format Error in a request frame
FMA	Fieldbus Management
FMS	Fieldbus Message Specification
GAP	Range of station addresses from this station to its successor in the logical token ring
IP	Invalid Parameter
IV	Invalid parameters in request
LE	Data area length too large (Up-/Download)
LLI	Lower Layer Interface
LSAP	Link Service Access Point identifies one FDL User in a particular station
L_sdu	Link_service_data_unit
NA	No Acknowledgement, no reaction from remote station
NC	Master parameter set not compatible
NE	Area at responder not available
NI	Function at responder not implemented
NO	Service in this state not possible
NR	No Response Data
OK	Service finished according to the rules
PDU	Protocol Data Unit

PHY	Physical Layer
PLC	Programmable Logic Controller
PNO	PROFIBUS Nutzerorganisation
R_d	Pulldown Resistor
RE	Format Error in a response frame
RR	Resources of the remote-FDL entity not sufficient or not avaiable
RS	Service or remote-address at remote-LSAP or remote-LSAP not activated; - remote-station is no DP-Station - remote-station is not yet ready for these functions - remote-station is associated with an other Requester - optional service not available
R_t	Termination Resistor
R_u	Pullup Resistor
RxD	Receive Data signal (PHY, RS-485)
SAP	Service Access Point
SC	Sequence Conflict; Function in the actual operation mode not allowed
SE	Sequence Error
TO	Function-Timeout expired
TxD	Transmit Data signal
UE	Remote DDLM/FDL interface error
VP	Voltage-Plus

4 Terms

Table 1. Terms

Access Protection	The master access to a DP-Slave is disabled for other DP-Masters.
Address-assignment-table	List in which, as an Example, the PLC-addresses are assigned to the decentraliced inputs and outputs.
Blocked data transfer	The data-range that should be transferred is transmitted in several blocks (parts).
Bracket data transfer	A data transmission is bracketed by a start-sequence and an end-sequence. Inside this sequence, the access of any other DP-Master is disabled.
Channel	Input- or output-data-range of a DP-Slave. Width: One bit up to a double word.
Channel related Diagnostic	Lowest level of slave specific diagnostic. The diagnostic information refers to a single input or output channel of the DP-Slave.
Composite-device	A device that provides PROFIBUS-FMS and PROFIBUS-DP services.
Configuration	In the start-up phase, the DP-Master (class 1) transmits the expected configuration for comparison with the real configuration to the DP-Slave.
Configuration error	If the DP-Slave detects a difference between the expected configuration and the real configuration, an error-flag is set in the diagnostic-information.
Coherent data range	The input- or output-data-range that always shall be transmitted coherently between the DP-Master and the DP-Slave.
Control Commands	Control commands will be transferred from the DP-Master to the DP-Slave. With the control commands it is possible to clear the outputs, or to freeze the inputs and/or outputs of a single or many DP-Slaves.
Control timers	- An adjustable control timer of the DP-Slave to detect a breakdown of the assigned DP-Master. - An adjustable control timer of the DP-Master (class 1) for monitoring the user data exchange mode with the corresponding DP-Slaves.

DDLM	The main task of the Direct Data Link Mapper (DDLM) is the mapping of the functions that are called at the interface to the User-Interface, on the FDL- and FMA1/2-services, see section "Protocol Architecture".
Decentralized Peripherals	Input- or Output-devices that are connected via a fast serial link with the central controller.
Default-Address	In case of the initial-clear of a DP-Slave, it changes its address to the address 126.
Device related Diagnostic	Highest level of the slave-specific diagnostic. The diagnostic information is related to the whole slave.
Diagnostic Buffer	A register that is able to store the diagnostic information temporarily.
Diagnostic Information Collection	A system-diagnostic information that is collected at the DP-Master (class 1).
Diagnostic Information	The status- or error-messages that are present at a DP-Master (class 1) or DP-Slave.
DP-Master (class 1)	The DP-Master that polls the assigned DP-Slaves and handles the user data exchange.
DP-Master (class 2)	The DP-Master that interacts as a configuration or diagnostic tool; usually a programming device.
Expected Configuration	Same structure as Real Configuration. During the start-up phase, the DP-Master (class 1) shall transfer this information to the DP-Slave for comparison.
Freeze inputs or outputs	An update of the inputs or outputs will be made. This state remains until the next control command will be received.
Grouping	DP-Slaves can be gathered into groups.
Identifier	For each input- and/or output-range of a DP-Slave an Identifier-byte is defined in the configuration phase.
Identifier related Diagnostic	Medium level of slave specific diagnostic. The diagnostic information is related to the input- or output-range for which a related Identifier-byte is defined in the configuration phase.
Master Parameter Set	The master parameter set includes the configuration and parametrization data of all DP-Slaves that are assigned to the corresponding DP-Master and the general bus parameters.

Mixed-Mode	PROFIBUS-FMS and PROFIBUS-DP devices are operating simultaneously on the same transmission medium.
Real Configuration	The amount and the width of the input- and output-data-ranges of the DP-Slave. Additionally, the description of the data consistency is included in the real configuration.
Reset	A hardware or software initiated resetting of a DP-Slave.
Scheduler	Controls the sequence of actions in the DP-Master (class 1).
User-Interface	The User-Interface represents in the layer model the interface to the user.
Vendor Identifier	This identifier specifies a DP-Slave type. It is administered and allocated by the PNO.

5 Requirements on the PROFIBUS-DP System

To switch from peripherals plugged in a controller to remote inputs/ outputs a fast data link is necessary not only for the exchange of process values, but also to transmit diagnostics, configuration data and parameters.
This application requires a communication system with short reaction times. Table 2 gives an overview of the requirements and features of PROFIBUS-DP system.

Table 2. Requirements and Features of PROFIBUS-DP

Requirement	Feature of PROFIBUS-DP
Short reaction time	Exchange of more than 1000 Inputs and Outputs with 32 devices in less than 10 ms.
Monomaster or Multi-master Operation	Hybrid media access according to PROFIBUS
Simple protocol, low cost communication interface	reduced functionality of PROFIBUS ASIC solution without microprocessor support
Proven transmission technology	Hamming Distance 4 guaranteed in a PROFIBUS system
Excellent diagnostic	Various diagnostics in Masters as well as in Slaves
Simple User-Interface	Basic set of parameters and configuration data
Use of existing cabling and test equipment	PROFIBUS FMS- and DP- devices can operate on one network. Same technology of transmission in all applications
Interoperability	Precise and complete definitions inclusive the definition of the system behaviour.

6 Behaviour of the System

6.1 General

In a typical remote I/O configuration single master configurations are mainly used to fulfil reaction time requirements. In lower speed applications multimaster configurations are also possible. PROFIBUS-DP uses the polling principle for communication (Master-Slave method). This means a DP-Slave station needs a Master-request to exchange information. The message transfer is organized in cycles. A message cycle mainly consists of a request-frame of the active station followed by the corresponding acknowledgement/response-frame of the addressed station. An exception to this is the global-control function for syncronization and coordination of several remote I/O stations.
A PROFIBUS-DP system consists of several communicating units. These may be DP-devices or devices working according to the FMS protocol of PROFIBUS.

Several station types can be identified:

- DP-Master (class 1)
- DP-Master (class 2)
- FMS-Master
- DP-Slave
- FMS-Slave

DP-Master (class 1):
This DP-Master controls several DP-Slaves according to a well defined algorithm. The DP-Master (class 1) uses the PROFIBUS-DP functions to communicate with DP-Slaves.
The DP-Master (class 1) polls the associated Slaves to submit data to its local user. The DP-Master (class 1) can communicate to a DP-Master (class 2) by a set of functions. This means a DP-Master (class 1) acts both as requester and responder.

DP-Master (class 2):
In a PROFIBUS-DP system a DP-Master (class 2) is a programmer or a management-device. A set of functions supports management and diagnostics of a complex DP-system.
The local user of the DP-Master (class 2) defines which operation has to be done depending on the kind of service required and which station has to be addressed.

FMS-Master:
This is a Master that communicates according to the rules of PROFIBUS using the FMS protocol.

DP-Slave:
A DP-Slave can be addressed by both types of DP-Master. This DP-Slave implements a defined set of responder functions.

FMS-Slave:
This defines a type of Slave that communicates according to the rules of PROFIBUS using the FMS protocol.

Before commissioning a DP-system, all stations shall have been assigned an unique address. In the case of DP-Slaves, this address can be set up via bus (see section "Change Station Address of a DP-Slave").
All DP-Slaves that have not yet been assigned an individual address, start with default address 126. Only one device with this address is allowed on the network at a certain time. It is not allowed to set a DP-Master address to the default address. A DP-Master (class 2) can access a Slave via that address and can assign a specific address to that device.
For security reasons, the DP-Master (class 1) shall not exchange I/O-data with a DP-Slave whose address is 126.
To exchange I/O data a DP-Master (class 1) shall have a valid master parameter set. This database consists of several sets of configuration data and parameters, each dedicated to a DP-Slave station. This set of data contains all necessary information (for the DP-system) to describe a DP-Slave. Additionally, the master parameter set includes the bus parameters as well as the address-assignment-table which assigns each individual remote I/O byte an unique address in the I/O space of the DP-Master's user.
If there exists a valid master parameter set in a DP-Master (class 1), this device starts to check whether the DP-Slaves dedicated to that DP-Master are present or not. After an appropriate answer the DP-Master will set the parameters and check the configuration of that DP-Slave. A DP-Slave will accept a check configuration request only from the Master who sets the parameters.
After submitting these two functions correctly, a DP-Master checks the status of the DP-Slave by reading the diagnostic-data. If the configuration check was successful and parametrization was correct, the DP-Master will enter the user data exchange mode.
A DP-Slave accepts data exchange requests only from the DP-Master which has previously submitted parameters and configuration.
A DP-Master (class 1) can send new parameter data to a DP-Slave without leaving the user data exchange mode.
Additionally, every DP-Master (class 2) can read diagnostic, inputs and outputs of every DP-Slave.
If there occurred a diagnostic event in the application process of a DP-Slave, this event is signalled by sending a response with high priority. The DP-Master has to read the diagnostics and to inform its local user.
A DP-Master (class 2) can take a DP-Slave under its control. In this case the DP-Slave will stop the data exchange with the DP-Master (class 1). The DP-Master (class 1) will recognize that and starts to read the diagnostics cyclically. It checks the field "Master-address" as long as there is a value not equal to invalid. If the DP-Master (class 2) has finished its communication with that DP-Slave, it will set the field "Master-address" to invalid. This event triggers the DP-Master (class 1) to gain control again. As described above it will send parameters and configuration first.

6.2 Synchronisation

The PROFIBUS-DP system has the capability to synchronize input- and output-data. These control commands are sent by means of the function Global_Control from a DP-Master to one, a group of, or all DP-Slaves simultaneously.
With each freeze control command the addressed DP-Slaves read their inputs into special input buffers. If a read command is issued by a DP-Master, the DP-Slave will send back the "frozen" input buffers ignoring any change in the recent past.
With each Sync control command the previously transmitted output values (see Data_Exchange) will be transferred to the output ports. The output values transferred with the following Data_Exchange functions will be stored and not transferred to the output ports till the next Sync control command follows or the Sync-Mode will be switched off.

The DP-Master (class 1) sends information about his own operation mode to his dedicated DP-Slaves for synchronisation. This will be done at every change of the operation mode of the DP-Master (class 1) or occures in time intervals wich can be parametrized.

6.3 Combination Devices

Besides the types of devices defined in section "Behaviour of the System", it is possible to build the following functional combination of devices:

- combination of DP-Master (class 1) and DP-Master (class 2)
- combination of DP-Master and FMS-Master
- combination of DP-Slave and FMS-Slave
- DP-Slave with DP-Master (class 1) responder functionality refered to DDLM- and User-Interface functions

7 Communication Model

7.1 General

PROFIBUS-DP not only describes communication functions, but also a fixed DP application. According to the ISO communication model, this functionality is placed in the user-interface. This DP application depends on the device-class (DP-Master (class 1), DP-Master (class 2) and DP-Slave). For efficiency reasons the DP-protocoll has no application layer (layer 7, application layer).
PROFIBUS-DP uses only the data transfer services of FDL and FMA1/2 as defined in PROFIBUS.
To offer a more comfortable layer 2-access, the user needs a direct data link mapper (DDLM) with predefined DP communication functions.

7.2 Protocol Architecture

This section describes PROFIBUS-DP in the environment of the ISO/OSI basic reference model, see DIN ISO 7498.

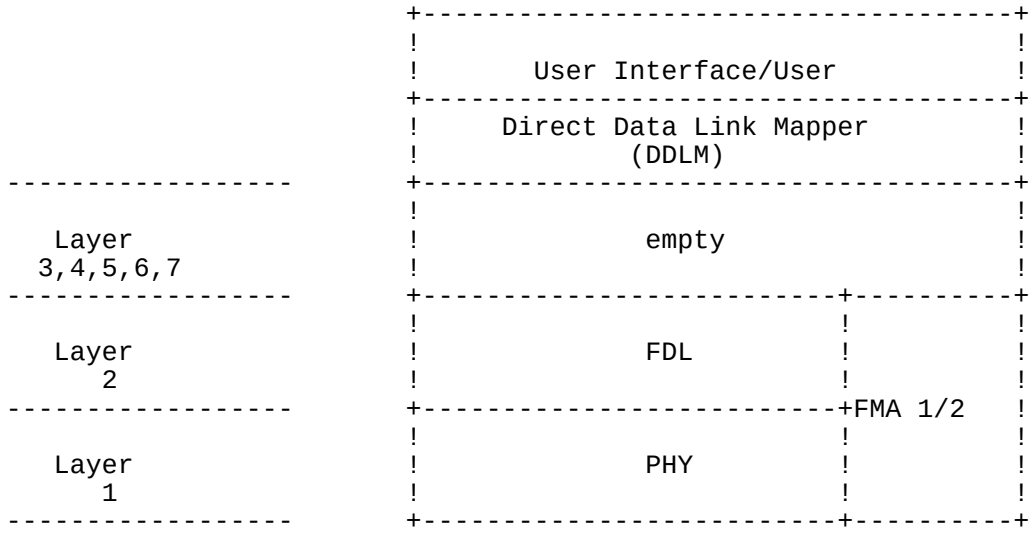


Figure 1. PROFIBUS-DP Layer Model

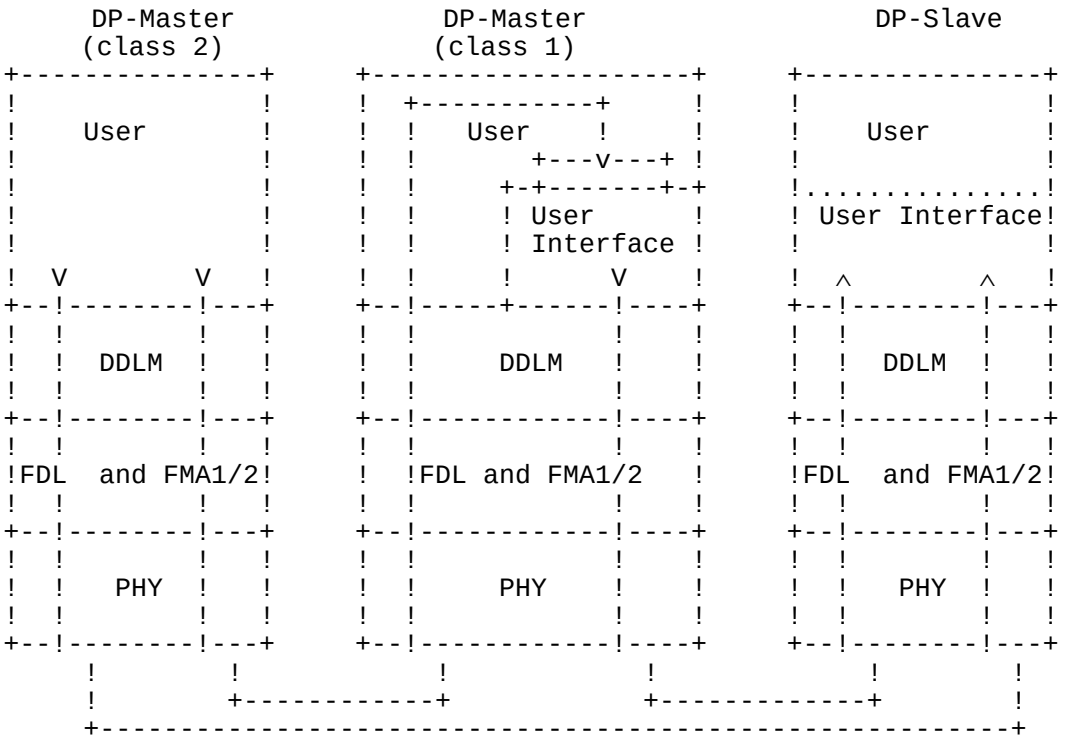


Figure 2. Communication Model

The application (user) of both DP-Master (class 1) and DP-Slave (e. g. PLC as Master and remote I/O as Slave) communicate via the User-Interface by using predefined DP applications.

The User-Interface of a DP-Master (class 1) implements the following Master-Slave application functions:

- read diagnostic information of DP-Slaves
- cyclic user data exchange mode
- parametrization and configuration checking
- submit control commands.

These functions are processed user independent. The interface between user and User-Interface consists of several service calls and a shared database.

Between a DP-Slave and a DP-Master (class 2) the following functions can be implemented additionally:

- read configuration of a DP-Slave
- read Input/Output-values
- address-assignment to DP-Slaves

A DP-Master (class 2) invokes the following functions of a DP-Master (class 1):

- read the DP-Master's (class 1) diagnostic information of the associated DP-Slaves
- upload and download of parameters
- activate bus parameters
- activate and deactivate DP-Slaves
- select the operating mode of a DP-Master (class 1).

These functions are processed by the user of a DP-Master (class 1).

The DDLM offers for the user or the user-interface in all three types of devices a comfortable access to the PROFIBUS Layer 2.

7.3 Communication Relationships

There are two types of communication in PROFIBUS-DP:

- one-to-one
- one-to-many (Multicast)

The communication depends on the kind of media access of the different stations:

- Master-Slave communication
  between DP-Master (class 1) and DP-Slaves
  between DP-Master (class 2) and DP-Slaves

- Master-Master communication
  between DP-Master (class 2) and DP-Master (class 1)

All types of communications operate connectionless.

In a PROFIBUS-DP system the initiator of a communication request (function) is always called the requester and the destination station is called the responder or receiver.
The initiator of a Master-Slave communication is always a DP-Master. The initiator of a Master-Master communication is always a DP-Master (class 2). No communication is defined between DP-Masters of the same class.

The figures 3 till 6 give an overview of the allowed types of communication.

All Master-Slave types of communication with the exception of DDLM_Global_Control are one-to-one communications.

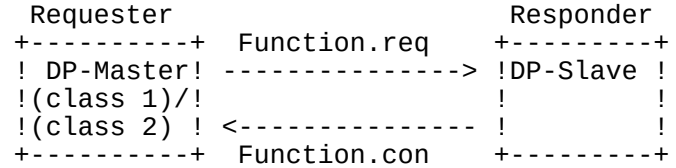


Figure 3. Master-Slave communication, one-to-one

The Master-Slave communication function DDLM_Global_Control is a Multicast communication. It is submitted from a DP-Master (class 1 or class 2) to associated DP-Slaves. This association was made during parametrization of the DP-Slave. Layer 2 access protection is activated at the beginning of the user

data exchange mode. DP-Slaves which are associated to a DP-Master can be devided into groups. Groups can be formed by using the group selector.

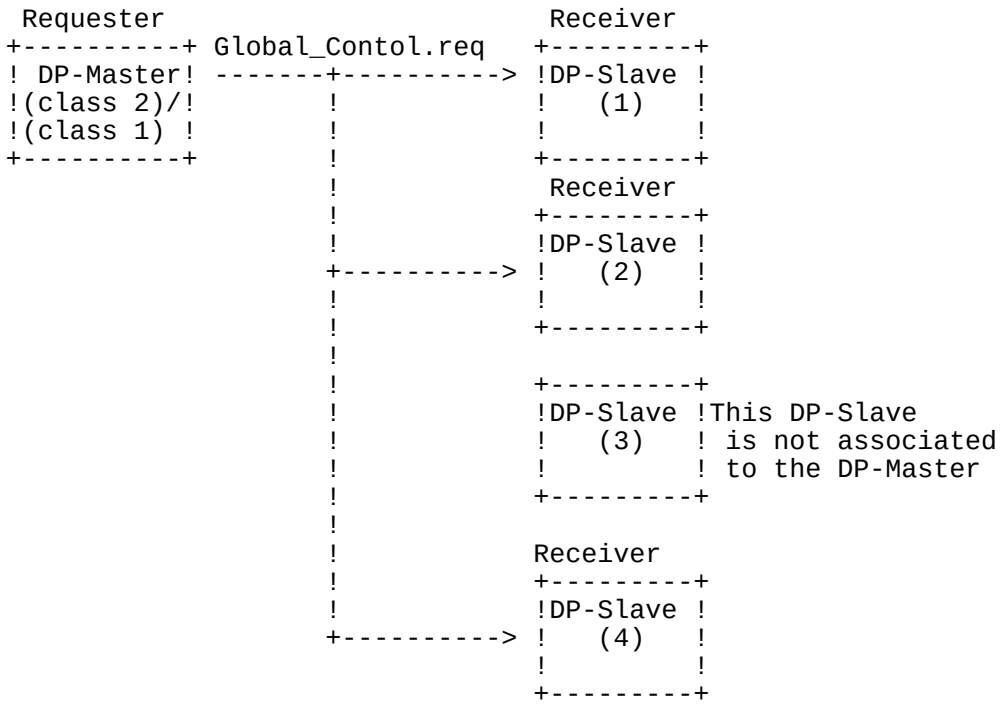


Figure 4. Master-Slave communication , one-to-many, Function DDLM_Global_Control

All Master-Master communication functions are mapped to a one-to-one communication.

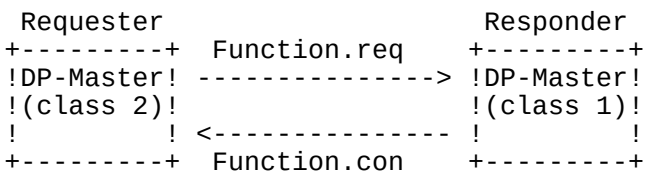


Figure 5. Master-Master communication, one-to-one

The Master-Master communication function DDLM_Act_Para_Brct can additionally mapped as a one-to-many communication.

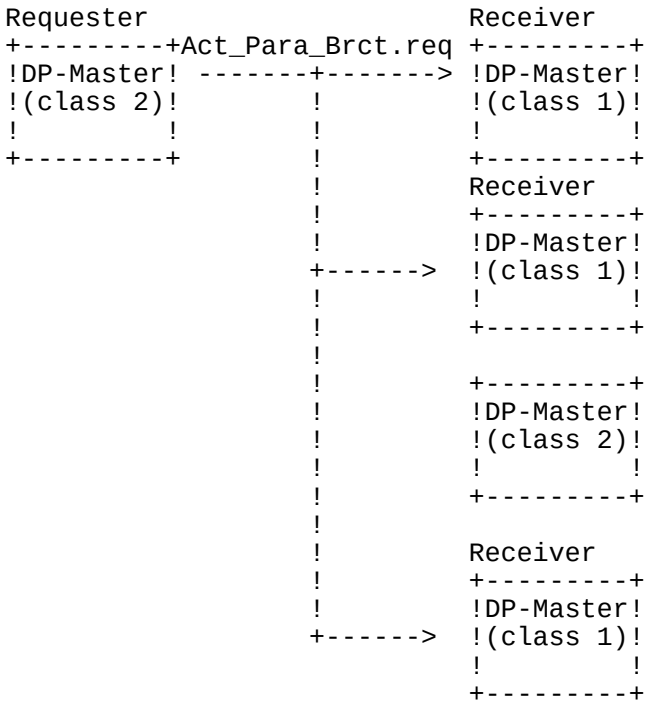


Figure 6. Master-Master communication, one-to-many, Function DDLM_Act_Para_Brct

7.4 Overview of Functions

PROFIBUS-DP offers the following basic functions:

- Master-Diagnostic read

Via this function a DP-Master's (class 1) diagnostic information about its associated DP-Slaves can be read. The diagnostic information consists of a global diagnostic overview and station diagnostics. A parameter switch selects the requested functionality.

- Parameter Up-/Download

These functions allow the transfer of parameter sets between two DP-Masters. Special parameter sets are for Example the bus parameters and the DP-Slave parameters. The second one is needed to set parameters and check the configuration of an associated DP-Slave.

- Activate bus parameter

This function activates a previously loaded set of bus parameter.

- Activate/Deactivate DP-Slaves

With this function a DP-Master can be forced to stop polling the DP-Slave or to start again to poll a DP-Slave.

- DP-Slave-Diagnostic information read

An application can read diagnostic information from a DP-Slave with this function.

- Data exchange of the inputs/outputs

This function is used for cyclic I/O-Data exchange between DP-Master (class 1) and associated DP-Slaves. The number of inputs and outputs is defined in the configuration data at startup of the DP-System.

- Set parameters of the DP-Slave

This function sets the parameters of the DP-Slaves at system startup, after a restart and in the user data exchange mode of a DP-System. There are some parameters that are unique in all DP-Slaves, the others are specific for each station.

- Check configuration of a DP-Slave

With this function the DP-Slave can check the configuration. The main purpose is to define the number and structure of inputs and outputs.

- Send control commands to DP-Slaves

A Master can send special control commands to one (single) or several (multicast) DP-Slaves. Different commands are distinguished by a parameter.

- Read configuration data of a DP-Slave

If a master has no configuration data of a DP-Slave it can read this information by means of that function.

- Read inputs and outputs of DP-Slaves

This function enables all DP-Masters to read the inputs and outputs of a DP-Slave while it is under control of a specific master.

- Change station address of a DP-Slave

The station-address can be set by a DP-Master during initialization.

7.5 Service Execution

This section shows the processing of PROFIBUS-DP functions according to well established description techniques. A function consists of a set of primitives. The primitives are adapted to the DP-Protocol architecture.
All DDLM functions are executed sequentially in a fixed order. Parallel or concurrent function requests are not allowed.

7.5.1 Master-Slave Communication

Communication requests are initiated at the User-Interface of a Master and will be sent to the DDLM by means of the function primitive Request (.req). The DDLM Acknowledgement will be received using the function-primitive (.con).
Depending on the type of DDLM function, four sequences of processing can be identified between User/User-Interface, DDLM and PROFIBUS layer 2. All Master-Slave communication functions are executed in one layer 2 message cycle (immediate response). This means that the data which has to be read of the DP-Slave will not be transferred with an indication-response sequence, but in advance with an Upd.req-primitive to layer 2.
The coding of the DDLM functions is implicit by using a defined set of layer 2 service access points. This coding information is sufficient for the DDLM in the case of Master-Slave communication.

The definitions of the possible Master-Slave functions can be found in section "DP-Master - DP-Slave Functions".

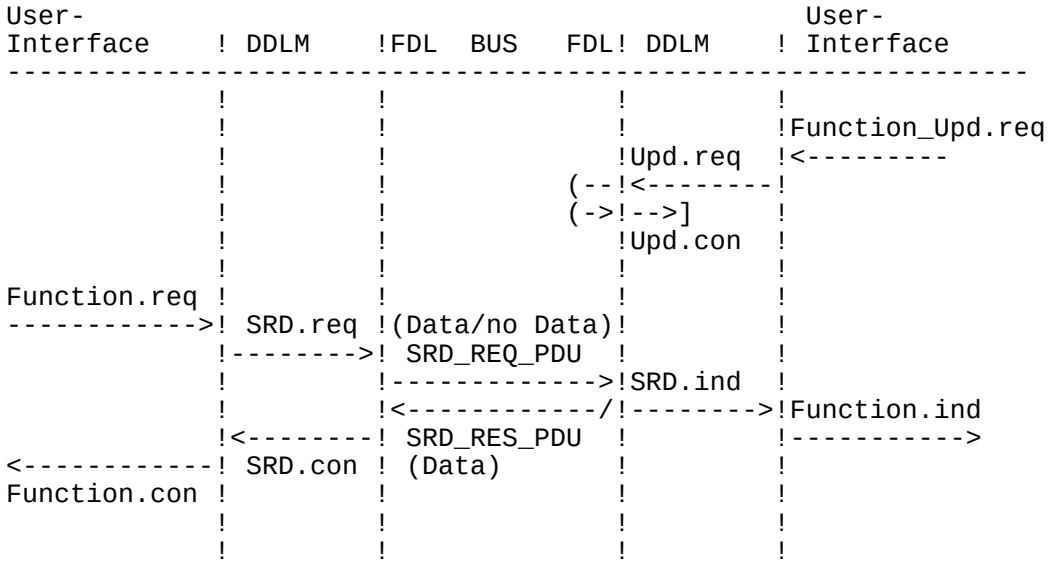


Figure 7. Processing of Master-Slave communication of the DDLM_functions DDLM_Data_Exchange and DDLM_Slave_Diag

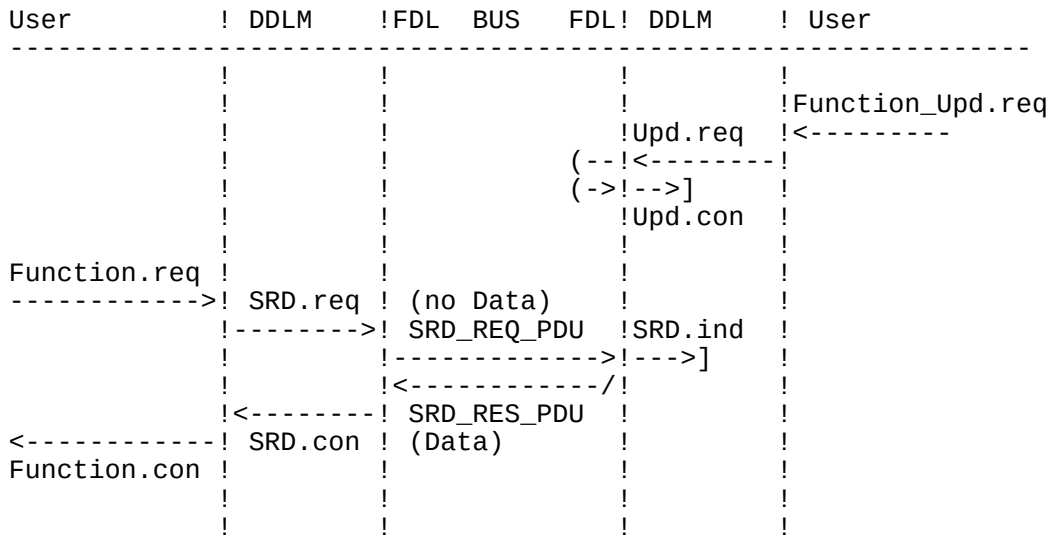


Figure 8. Processing of Master-Slave communication of the DDLM_functions RD_Inp, RD_Outp and Get_Cfg

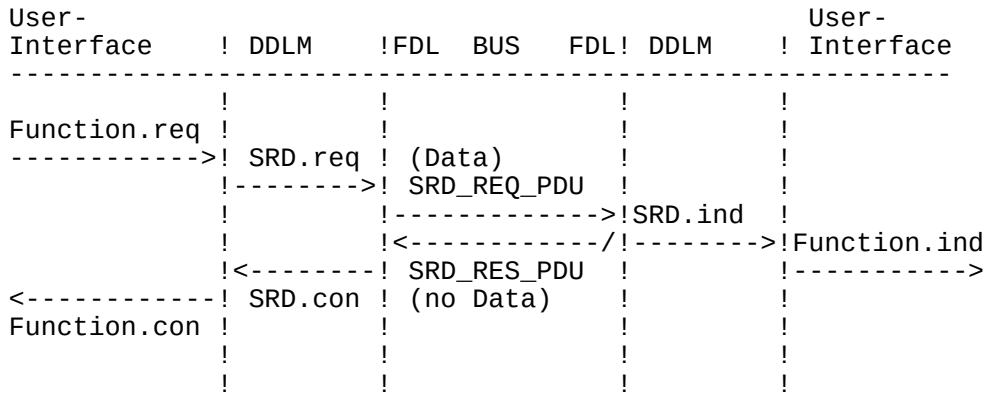


Figure 9. Processing of Master-Slave communication of the DDLM_functions Set_Prm, Chk_Cfg and Set_Slave_Add

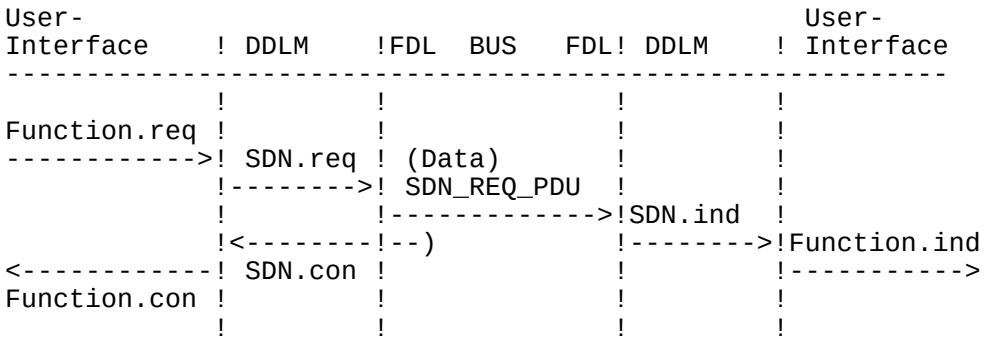


Figure 10. Processing of Master-Slave communication of the DDLM_function DDLM_Global_Control

7.5.2 Master-Master Communication

Communication requests are initiated by the User Interface of a DP-Master (class 2) and will be sent to the DDLM with the function primitive Request (.req). The DDLM acknowledgement will be received with the function primitive (.con).
Depending on the type of DDLM function, two types of sequences of processing can be identified between User and DDLM. The Master-Master communication functions will be executed in one or more layer 2 message cycles. A DP-Master (class 1) processes all communication-functions with the exception of DDLM_Act_Para_Brct with an Indication-Response-Sequence.
The Coding of the DDLM-Functions is accomplished through a defined layer 2 service access point. Additionally, the Master-Master communication needs more coding information. This coding will be placed by the DDLM in the first octet of the Data-Unit of layer 2.

An overview of possible Master-Master functions can be found in section "DP-Master - DP-Master Functions".

Instead of these Master-Master DDLM communication functions, FMS-services according to PROFIBUS Application Layer Service Definitions can be used. This applies for devices which use PROFIBUS-DP and PROFIBUS-FMS simultaneously (see annex 2-B.1).

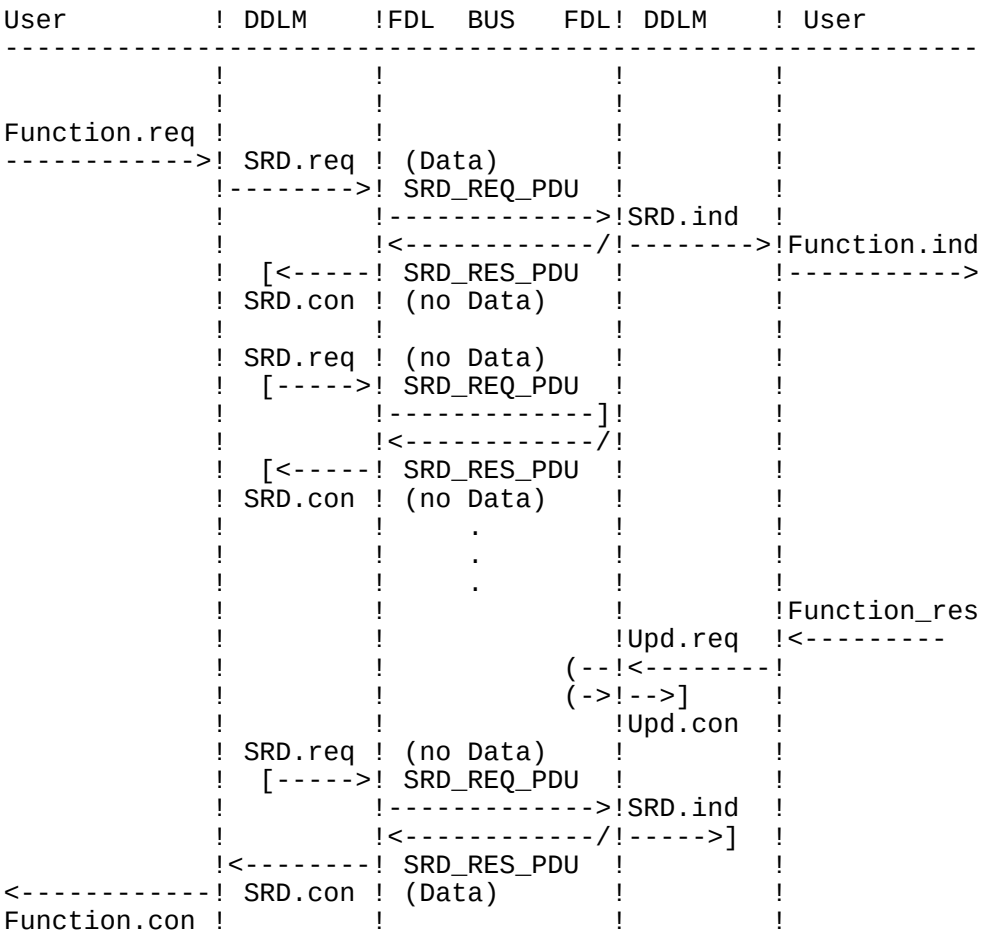


Figure 11. Processing of all Master-Master communications except DDLM_Act_Para_Brct

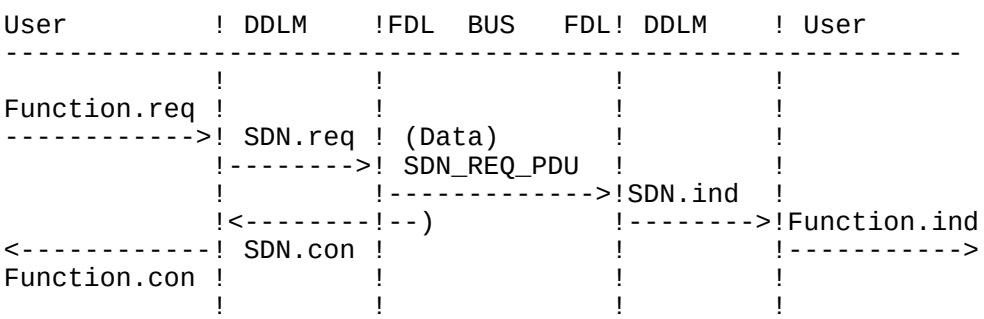


Figure 12. Processing of the Master-Master communication DDLM_Act_Para_Brct

8 Medium Access and Transmission Protocol

8.1 General

The medium access and transmission protocol has to be implemented according to the PROFIBUS Specification. PROFIBUS-DP distinguishes between active and passive stations. The active stations form a virtual Token-Ring.
Passive stations do not participate in token passing and are thus not members of the virtual token ring. They shall be polled by a Master in a Master-Slave communication cycle.
In PROFIBUS-DP a DP-Slave is associated with a distinct DP-Master. The DP-Master (class 1) is polling the associated DP-Slaves cyclically according to the master's data base.
The DP-Master (class 2) communicates with DP-Slaves acyclically.
The FMS-Master is polling the associated FMS-Slaves according to its Poll List cyclically or acyclically.

The following configurations are possible:

- PROFIBUS-DP System (Monomaster)

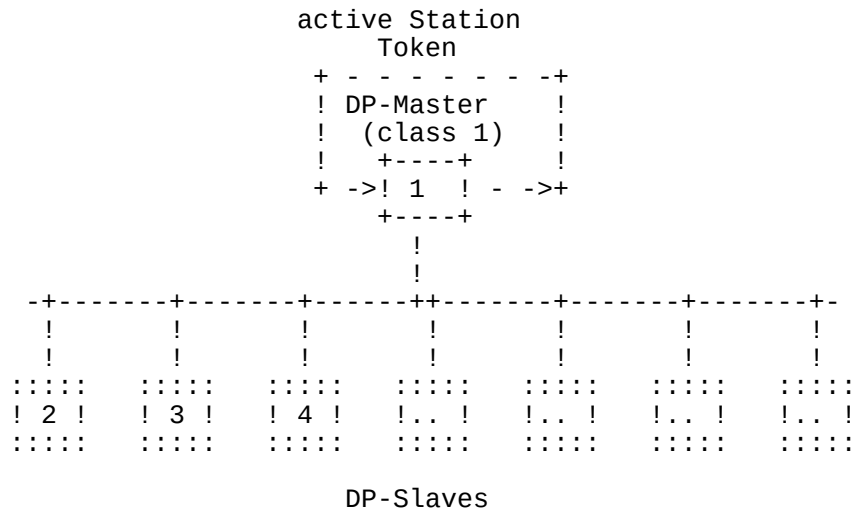
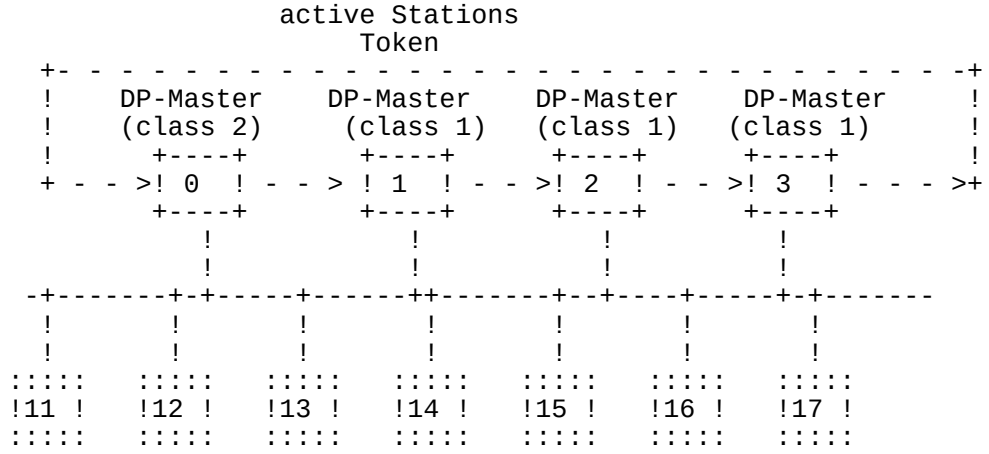


Figure 13. Monomaster PROFIBUS-DP System

Figure 13 shows a PROFIBUS-DP Monomaster-System. The system consists of a DP-Master (class 1) and up to 125 DP-Slaves.
According to the Data Link Layer Service Protocol Specifications of PROFIBUS the DP-Master (class 1) passes the Token regularly (if there is no request or the T_{TR} expired) to itself.

- PROFIBUS-DP System (Multimaster)



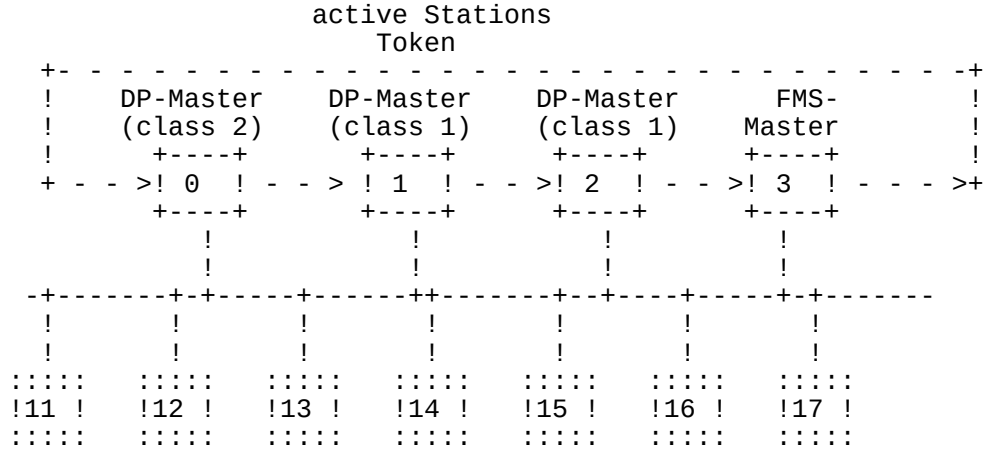
Possible associations of DP-Slaves to the Master:

DP-Slave:	11,12,13	DP-Master:	1
DP-Slave:	14,15	DP-Master:	2
DP-Slave:	16,17	DP-Master:	3

Figure 14. Multimaster PROFIBUS-DP System

Figure 14 shows a PROFIBUS-DP Multimaster system configuration. The association of DP-Slaves to the DP-Master (class 1) is defined during system configuration and will be stored in the master's parameter set. Up to 126 devices are possible on one network.

- mixed PROFIBUS System



Possible association of Slaves to the Master:

DP-Slave: 11,12,13	DP-Master: 1
DP-Slave: 14,15	DP-Master: 2
FMS-Slave: 16,17	FMS-Master: 3

Figure 15. Mixed PROFIBUS-System

The structures in Figure 14 and 15 show the general outline of a Multi-Master-System. The Token is passed from active station to active station. The Master that currently holds the token is able to access the media as a requester.
In a mixed PROFIBUS-System, PROFIBUS-DP devices and PROFIBUS-FMS stations can use the same media. They are compatible at the physical and media access layer, but they can not exchange information at application level.
Depending on the requirements, several FMS- and/or DP-Masters are allowed. Up to 126 devices are possible on one network.

For performance reasons a PROFIBUS-DP system should not consist of more than 3 DP-Masters.

8.2 Token Rotation Time

A very important fact for a DP-Master (class 1) connected to field devices is a fixed cycle time. A DP-Master (class 1) has to have a Target Rotation Time (T_{TR}) large enough, so that it can poll each associated DP-Slave once per token cycle.
In a multimaster system the cycle time of each DP-Master (class 1) has to be extended by the token hold time of all other masters.
To maintain the cycle time of the DP-system in a specific DP-application, a DP-Master (class 1) should have a sufficient Target Rotation Time (T_{TR}) for a complete poll cycle. To reach this goal, it is necessary to prioritize the DP-Master (class 1) towards the other Masters. Thus, the T_{TR} is set in a way that the DP-Masters (class 1) add the time of a complete poll cycle to the T_{TR} of the other masters in the ring.

Calculation:

for FMS-Masters and DP-Masters (class 2):

$$T_{TR(FMS)} > \Sigma T_{SLP} \quad (1)$$

for DP-Masters (class 1):

$$T_{TR(DP)} = T_{SLP(TS)} + T_{TR(FMS)} + T_{ADD} \quad (2)$$

$$T_{ADD} = (T_{ID1} + T_{SL} + 2 \cdot 255 \cdot 11 \cdot t_{Bit}) \cdot (Max\_Retry\_Limit + 1) \quad (3)$$

Legend:
$T_{TR(FMS)}$ = FMS- and DP-Masters (class 2) Target Rotation Time
ΣT_{SLP} = Sum of the poll cycles of all DP-Masters (class 1)
$T_{SLP(TS)}$ = Poll cycle of the DP-Master (class 1) "TS"
$T_{TR(DP)}$ = the DP-Master's (class 1) Target Rotation Time
T_{ADD} = Safety margin for message retransmissions

The bus parameters T_{ID1} , T_{SL} , t_{Bit} , Max_Retry_Limit are described the Data Link Layer Service Protocol Specifications.

8.3 Priorities

The PROFIBUS-DP protocol uses high and low priority messages. All request messages are sent with high priority (Master-Slave-communication). If there is new diagnostic information in a DP-Slave the response messages in the user data exchange mode will be replied with high priority. In all other cases, the replies are sent as low priority messages.

8.4 Control Intervals

In industrial control systems there is a need for checking the correct functioning of each individual part. In a DP system a media failure or a defect in a DP-Master shall not cause errors in the peripheral system. In the above mentioned cases, a DP-Slave shall switch to a safe state. Additionally, the user of a DP-Master (class 1) has to be informed about transmission breakdowns after a certain amount of time.

- Control Interval at the DP-Slave:

This Timer (watchdog control), retriggered by received Master-requests, will set the outputs of a DP-Slave to the safe state after expiration of the timer.
The definition of the watchdog control time can be found in section "Send Parameter Data".

- Control Interval at the DP-Master (class 1):

The data transfer between DP-Master (class 1) and its associated DP-Slaves will be controlled by the User-Interface. By this means a master checks the data transfer to its DP-Slaves.
Additionally the Data_Control_Time defines the time in which a DP-Master (class 1) indicates his operation mode to the dedicated DP-Slaves.
The master parameter set contains the Data_Control_Time.

Rule:

$$T_{WD} > T_{TR} \quad (4)$$

$$\text{Data Control Time} \geq 6 \cdot T_{WD} \quad (5)$$

8.5 System Reaction Time

The system reaction time depends mainly on the number of stations, the number of bytes per DP-Slave and the number and types of masters in the system.
Furthermore, the maximum reaction time (max T_{SDR}) of the individual Stations will influence the system reaction time.
A typical DP-Station will have a max. T_{SDR} according to Table 4.

The calculation of the system reaction time is straightforward in a monomaster system. The formula below will predict the system behaviour.
To maintain a fast reaction time, the bus traffic shall not have gaps between messages.

The **theoretical** system reaction time is calculated as follows:

theoretical system reaction time =

$$[\text{Token} + \text{GAP-Request} + \text{Number Stations} \cdot \text{Offset} + \text{Number I/O-Bytes} \cdot 11 + T_{SM}] \cdot t_{Bit} \quad (6)$$

If the minimal values according to Table 3 can be applied, at a transmission speed of 1500 kBit/s (all stations have Inputs and Outputs) the cycle time equals to:

$$= [70 + 403 + \text{Number Stations} \cdot 246 + \text{Number I/O-Bytes} \cdot 11 + T_{SM}] \cdot t_{Bit} \quad (7)$$

The formula above assumes a mono-master-system. The DP-Master sends out the Token to itself. min T_{SDR} is set to 11 t_{BIT}, T_{ID1} / T_{ID2} is set to 37 t_{BIT}.

time for Token : $T_{ID1}+T_{Token} = (37+33)t_{BIT} = 70t_{BIT}$
time for GAP : $T_{ID1}+T_{SD1}+T_{SLOT} = (37+66+300)t_{BIT} = 403t_{BIT}$
time for Offset: $T_{ID1}+2 \cdot T_{SD2\_R}+\min T_{SDR} = (37+198+11)t_{BIT} = 246t_{BIT}$
time for T_{SM} : $= 1t_{BIT}$

Legend:

T_{Token}	Time to transmit a token-frame
T_{SD1}	Time to transmit a frame with Start Delimiter SD1
T_{SD2_R}	Time to transmit a frame with Start Delimiter SD2 (without Lsdu)

The bus parameters T_{ID1}, T_{SL}, t_{Bit}, T_{SM} are described in the Data Link Layer Service Protocol Specifications.

For communication networks with more than one master this calculation is complex. In this case, the individual token-hold-time of each DP-Master shall be part of the calculation.
Part 4 gives a basic rule for the token rotation time.

8.6 Frame Formats
The frame format of PROFIBUS-DP is according to the Data Link Layer Service Protocol Specifications.

8.7 Address Extension

The PROFIBUS-DP protocol uses service access points (SAP) of FDL as a basic function code.

The data exchange function has no address-extension (NIL SAP) for efficiency reasons.
The section "Direct-Data-Link-Mapper" gives the assignment of service access points to the functions.

Frames with address extension "Segment" (Bridge SEG Addresses) are not allowed for PROFIBUS-DP. Frames received with segment address-extensions are ignored in a DP station.

8.8 Bus Parameter

The following tables give an overview of the most important bus parameters and their values in a monomaster system.

Table 3. Bus Parameter/Reaction times for a DP-Master

Baud rate in kbit/s	up to 187,5	500	1500
T_{RDY} (t_{Bit})	≤11	≤11	≤11
T_{SDI} (t_{Bit})	≤80	≤180	≤280
Default			
T_{SL} (t_{Bit})	100	200	300
min T_{SDR} (t_{Bit})	11	11	11
max T_{SDR} (t_{Bit})	60	100	150
T_{SET} (t_{Bit})	1	1	1
T_{QUI} (t_{Bit})	0	0	0
G	100	100	100
HSA	126	126	126
max_retry_limit	1	1	1

For a DP-Master it is necessary to reach the values for T_{RDY} and T_{SDI} as shown in Table 3. The allowed scale of the parameters T_{RDY} and T_{SDI} are not fixed by this table. In a multimaster environment or in a mixed operation with FMS-Masters some bus parameters may have to be set to higher values. Especially the slot time may be extended because of token passing. Each master shall have the chance to receive the token and if necessary react (send a request or token) properly.

Table 4. Bus Parameter/Reaction times for a DP-Slave

Baud rate in kbit/s	up to 187,5	500	1500
max T_{SDR} (t_{Bit})	≤60	≤100	≤150
Default min T_{SDR} (t_{Bit})	11	11	11

For a DP-Slave it is not allowed to exceed the values of max T_{SDR} as shown in Table 4. The allowed scale of the parameter max T_{SDR} is not fixed by this table. In a multimaster environment or in a mixed operation with FMS it is possile that max T_{SDR} has to be set to higher values.
Another performance feature for a DP-Slave implementation is the time Min_Slave_Interval (see section "Data Interface"). Each slave implementation has to ensure that the Min_Slave_Interval reaches the smallest value that is possible. This means, that in a DP system with more than ten stations the Min_Slave_Interval shall not be the dominant factor for the cycle time.

Additionally, a DP-Slave shall implement a time T_{WD} (see section "Send Parameter Data").

8.9 Statistic Counters

To install and maintain a PROFIBUS system, a DP-Master may **optionally** implement the following statistic counter (FDL Variables). These counters shall be treated pairwise:

- Counter for sent frames (Frame_sent_count), without SDN-, FDL-Status- and Token frames.
- Counter for sent frames with no response or erroneous response (Error_count), except SDN-, FDL-Status- and Token frames.

These counters are maintained for each individual station.

9 Interface between DDLM and User-Interface

This section gives a detailed specification of DDLM functions and primitives which are provided on the interface between DDLM and User-Interface.

9.1 Device Specific Function Reference

Table 5 gives an overview of the functions which shall be implemented in a DP-Slave and in a DP-Master. Some functions are additionally possible. Requester and responder roles in function execution are given separatly.

Table 5. Attachment of the functions to DP-Masters and DP-Slaves

function	DP-Slave		DP-Master (class 1)		DP-Master (class 2)	
	Req	Res	Req	Res	Req	Res
Data_Exchange	-	M	M	-	O	-
RD_Inp	-	M	-	-	O	-
RD_Outp	-	M	-	-	O	-
Slave_Diag	-	M	M	-	O	-
Set_Prm	-	M	M	-	O	-
Chk_Cfg	-	M	M	-	O	-
Get_Cfg	-	M	-	-	O	-
Global_Control	-	M	M	-	O	-
Set_Slave_Add	-	O	-	-	O	-
Get_Master_Diag	-	-	-	M	O	-
Start_Seq	-	-	-	O	O	-
Download	-	-	-	O	O	-
Upload	-	-	-	O	O	-
End_Seq	-	-	-	O	O	-
Act_Para_Brct	-	-	-	O	O	-
Act_Param	-	-	-	O	O	-

Req = Requester
Res = Responder/Receiver
M = Mandatory
O = Optional

9.2 Description Format of DDLM-Function Calls

This section describes the general outline of the function-calls between User Interface and DDLM.
The necessary function parameters are given in a formal description as a table. The way to implement the functions is not defined in this specification.
All parameter values shall have a defined value at execution time. The format and the number of request or response parameters depends on the different functions.

The first column specifies the name of the parameter. The following columns are reserved for function primitives. It is marked whether or not a parameter is needed for a specific primitive.

The parameter may be structured. The subparameters and their belonging to a parameter is represented by indenting their names by 2 characters.
The information whether an optional parameter is used or which parameter of a selection has to be selected in a specific case is not mentioned in these tables.
However, the information shall be present at the DDLM/FDL- and DDLM/FMA1/2-Interfaces.

Representation of functions and their parameters.

Table 6. Representation of the Parameters of Function Calls

Parameter Name	.req	.ind	_Upd .req *)	.con	.res
Requestparameter1	M		M		
Parameter_A	S		S		
Parameter_B	S		S		
Requestparameter2	U		U		
Status				M	
Responseparameter1		M		M	
Responseparameter2		C		C	

*) In the place of the _Upd.req primitive the .res primitive is possible.

meaning:
- .req Request function primitive
- .ind Indication function primitive
- _Upd.req Update Request function primitive
- .con Confirmation function primitive
- .res Response function primitive
- M Parameter is mandatory
- U Parameter is an user option; can be used or omitted
- S Parameter is selected, there are several alternatives
- C the existence of the parameter depends of the value of a second parameter

The function acknowledgements (.res and .con) contain a status parameter, the possible values are defined below:

Table 7. Status Values of the Functions

Value	Meaning	Source
OK	Acknowledgement positive	
IV	Invalid parameters in request	Local
NO	Service in this state not possible	Local
DS	Local-FDL/PHY Entity is not in logical token-ring or disconnected from line	Local FDL
NA	Negative ack,no reaction from remote station	Local FDL
RS	Service or remote-address at remote-LSAP or remote-LSAP not activated; - remote-station is no DP-Station - remote-station is not yet ready for these functions - remote-station is associated with an other Requester - optional service not available	Remote FDL
RR	Resources of the remote-FDL Entity not sufficient or not available	Remote FDL
UE	Remote-DDLM/FDL interface error	Remote FDL
NR	No Response data	Remote FDL
TO	Function-Timeout expired	Local DDLM
FE	Format-Error in a Request-frame	Remote DDLM
RE	Format-Error in a Response-frame	Local DDLM
LE	Data-block-length too large(Up-/Download)	Remote User
NI	Function not implemented	Remote User
EA	Area too large (Up-/Download)	Remote User
AD	Access denied	Remote User
IP	Invalid Parameter	Remote User
SC	Sequence Conflict	Remote User
SE	Sequence Error	Remote DDLM
NE	Area non-existent	Remote User
DI	Data Incomplete	Remote User
NC	Master parameter set not compatible	Remote User

9.3 DP-Master - DP-Slave Functions

9.3.1 Read DP-Slave Diagnostic Information

The Data_Unit contains the diagnostic information. If a bit has been set this means that the event which is linked with this bit has occurred. The diagnostic information consists of a standard diagnostic information (octet 1 to octet 6) and an extended diagnostic information (Ext_Diag_Data).

Table 8. DDLM_Slave_Diag

Parameter Name	.req	.ind	_Upd .req	.con
Rem_Add	M			M
Req_Add		M		
Status				M
Diag_Data			M	C

Rem_Add:

The parameter Rem_Add (Remote_Address) specifies the FDL-Address of the DP-Slave from which the diagnostic information are requested.
Type: Unsigned8
Range: 0 to 126

Req_Add:

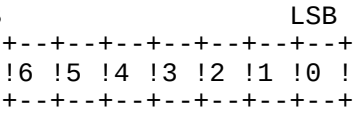
This parameter contains the station address of the requester. With the help of this address it is determined if the watchdog control shall be triggered.
Type: Unsigned8
Range: 0 to 125

Status:

The parameter Status indicates success or failure of this function.
Possible Values: OK,DS,NA,RS,UE,NR,RE

Diag_Data:
Type: Octet String
Length: 6 to 32 (extendible to 244, see section "Restrictions")

Octet 1: Station_status_1



The individual bits have the following meaning:

Bit 7: Diag.Master_Lock
The DP-Slave has been parameterized from another master. This bit is set by the DP-Master (class 1), if the address in octet 4 is different from 255 and different from the own address. The DP-Slave sets this bit to zero.

Bit 6: Diag.Prm_Fault
This bit is set by the DP-Slave if the last parameter frame was faulty, e. g. wrong length, wrong Ident_Number, invalid parameters.

Bit 5: Diag.Invalid_Slave_Response
This bit is set by the DP-Master as soon as receiving a not plausible response from an addressed DP-Slave. The DP-Slave sets this bit to zero.

Bit 4: Diag.Not_Supported
This bit is set by the DP-Slave as soon as a function was requested which is not supported from this DP-Slave.

Bit 3: Diag.Ext_Diag
This bit is set by the DP-Slave. It indicates that a diagnostic entry exists in the slave specific diagnostic area (Ext_Diag_Data) if the bit is set to one. If the bit is set to zero a status message can exist in the slave specific diagnostic area (Ext_Diag_Data). The meaning of this status message depends on the application and will not be fixed in this specification.

Bit 2: Diag.Cfg_Fault
This bit is set by the DP-Slave as soon as the last received configuration data from the DP-master are different from these which the DP-Slave has determined.

Bit 1: Diag.Station_Not_Ready
This bit is set by the DP-Slave if the DP-Slave is not yet ready for data transfer.

Bit 0: Diag.Station_Non_Existent
This bit is set by the DP-Master if the respective DP-Slave can not be reached over the line. If this bit is set the diagnostic bits contains the state of the last diagnostic message or the initial value. The DP-Slave sets this bit to zero.

Octet 2: Station_status_2

	MSB						LSB
Bit-No	!7	!6	!5	!4	!3	!2	!1	!0

The individual bits have the following meaning:

Bit 7: Diag.Deactivated
This bit is set by the DP-Master as soon as the DP-Slave has been marked inactive within the DP-Slave parameter set and has been removed from cyclic processing. The DP-Slave sets this bit always to zero.

Bit 6: reserved

Bit 5: Diag.Sync_Mode
This bit is set by the DP-Slave as soon as the respective DP-Slave has received the Sync control command.

Bit 4: Diag.Freeze_Mode
This bit is set by the DP-Slave as soon as the respective DP-Slave has received the Freeze control command.

Bit 3: Diag.WD_On (Watchdog on)
This bit is set by the DP-Slave as soon as his watchdog control has been activated.

Bit 2: This bit is set to 1 by the DP-Slave.

Bit 1: Diag.Stat_Diag (static diagnostics)
If the DP-Slave sets this bit the DP-Master shall fetch diagnostic informations as long as this bit is reset again. For Example, the DP-Slave sets this bit if it is not able to provide valid user data.

Bit 0: Diag.Prm_Req
If the DP-Slave sets this bit the respective DP-Slave shall be reparameterized and reconfigured. The bit remains set until parameterization is finished. This bit is set by the DP-Slave.

If bit 1 and bit 0 are set, bit 0 has the higher priority.

Octet 3: Station_status_3

	MSB						LSB
Bit-No	!7	!6	!5	!4	!3	!2	!1	!0

The individual bits have the following meaning:

Bit 7: Diag.Ext_Diag_Overflow
If this bit is set there exists more diagnostic information than specified in Ext_Diag_Data. For Example, the DP-Slave sets this bit if there are more channel diagnostics than the DP-Slave can enter in its send buffer; or the DP-Master sets this bit if the DP-Slave sends more diagnostic information than the DP-Master can enter in its diagnostic buffer.

Bit 0 to 6: reserved

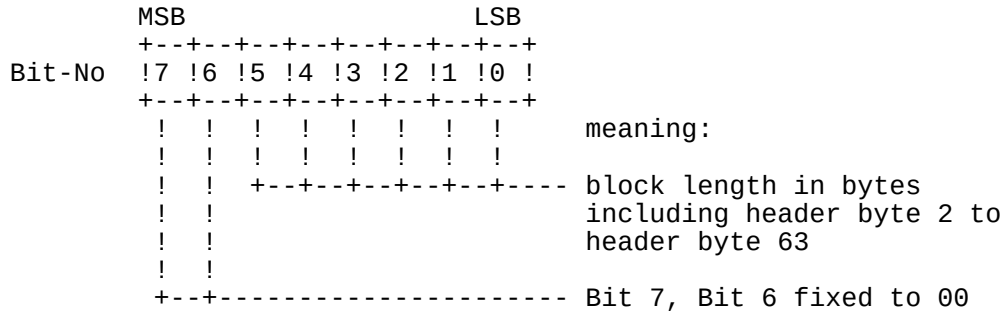
Octet 4: Diag.Master_Add
In this octet the address of the DP-Master is entered which has parameterized this DP-Slave. If none of the DP-Masters has parameterized the DP-Slave then the DP-Slave inserts the address 255 in this octet.

Octet 5 to 6 (unsigned16): Ident_Number
The manufacturer identifier is given for a DP-Device. This identifier can be used on the one hand for verification purpose and on the other hand for exact identification.

Octet 7 to 32: Ext_Diag_Data (extendible to 244, see section "Restrictions")
In this area the DP-Slave can enter its specific diagnostic. It is defined a block structure with one header byte each for the device related diagnostic and the identifier related diagnostic:

Device related diagnostic:

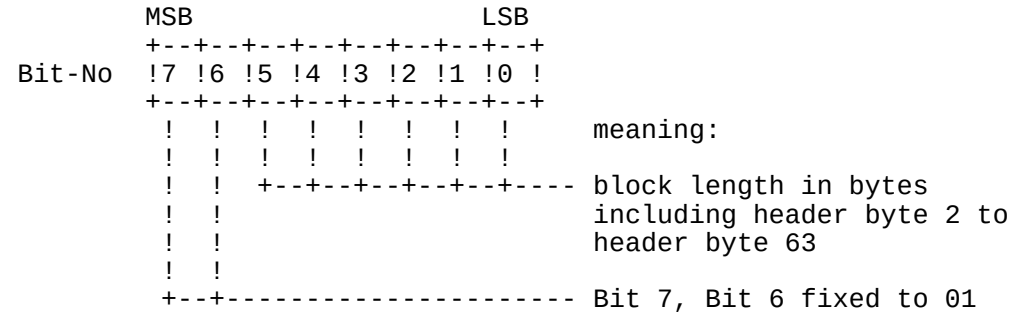
Header byte:



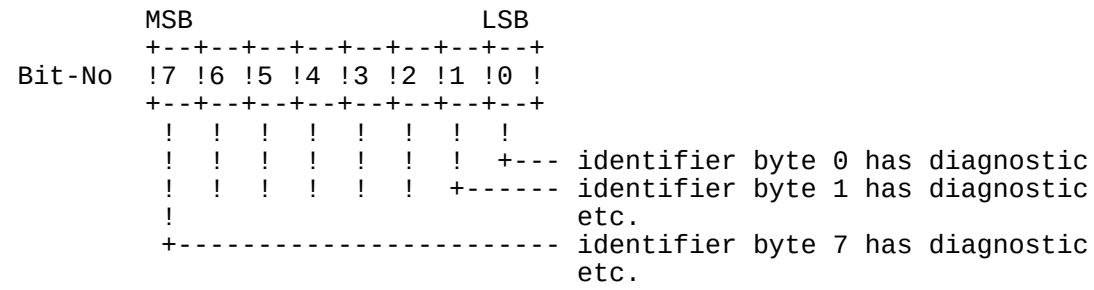
In this block general diagnostic information is entered as for Example overtemperature, undervoltage or overvoltage. The coding is defined device specific.
For further interpretation the Ident_Number shall be used.

Identifier related (module) diagnostic:
For each used identifier byte at the configuration one bit is reserved. It is padded to byte limits. The bits which are not configured shall be set to zero. A set bit means that in this I/O area diagnostic is pending.

Header byte:

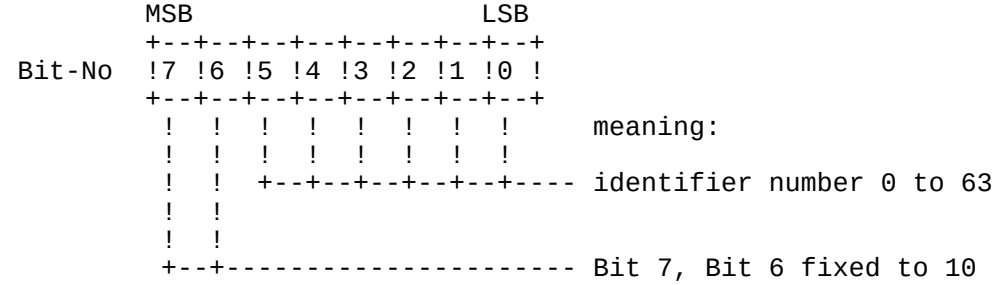


Bit structure for identifier related diagnostic:

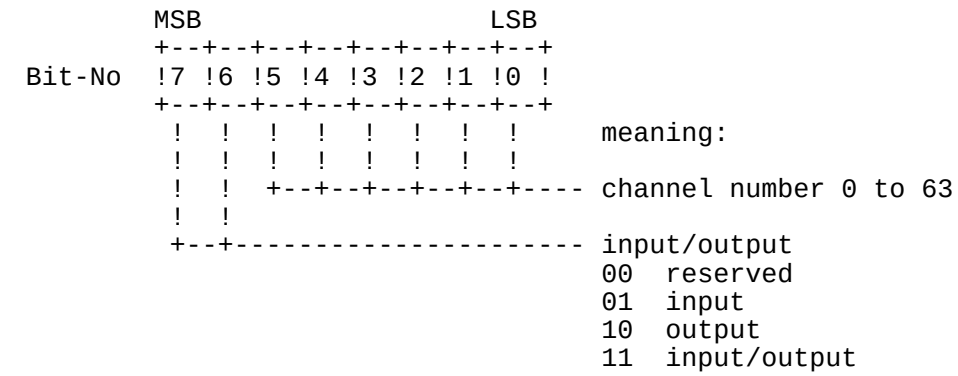


Channel related diagnostic:
In this block the diagnosed channels and the diagnostic reason are entered in turn. The length per entry is 3 octets.

Octet 1: Identifier number

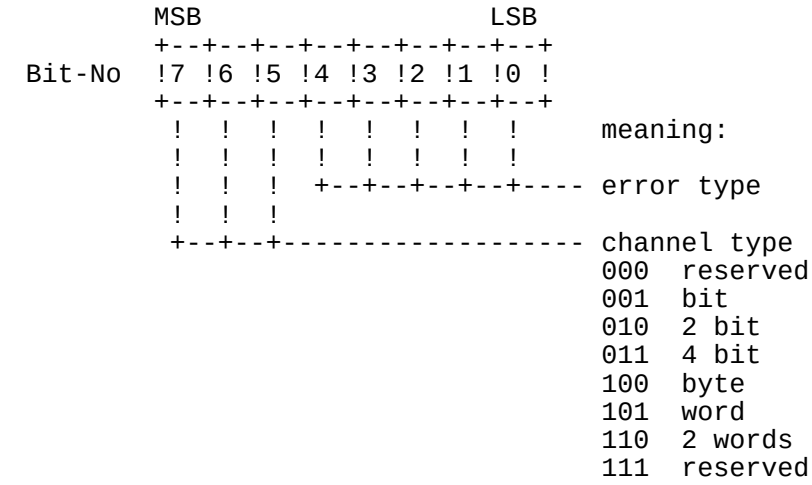


Octet 2: Channel number



For identifier bytes which contain both input and output, the direction of the diagnosed channel is indicated in bit 7 and bit 6 of the channel number.

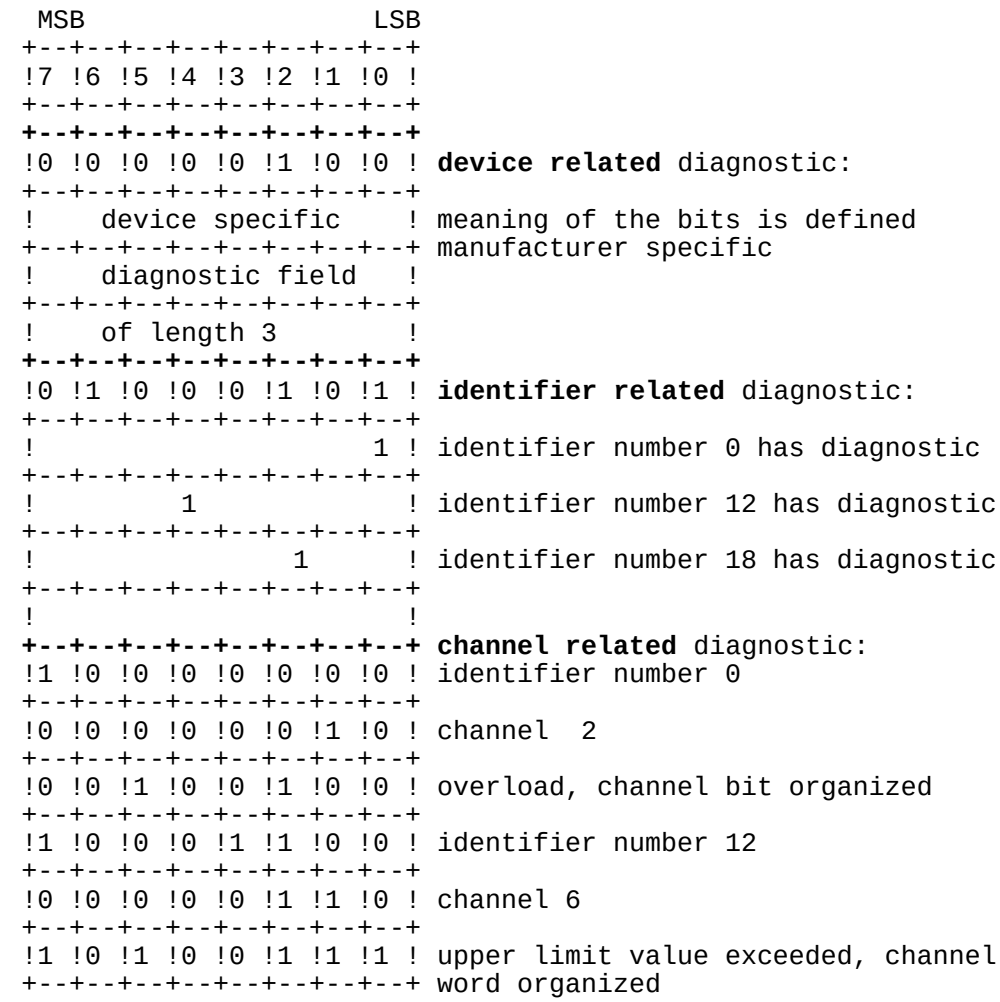
Octet 3: Type of diagnostic



Error type:

0	reserved
1	short circuit
2	undervoltage
3	overvoltage
4	overload
5	overtemperature
6	line break
7	upper limit value exceeded
8	lower limit value exceeded
9	error
10	reserved
.	
15	reserved
16	manufacturer specific
.	
31	manufacturer specific

Example: Structure of a diagnostic according to the pattern above



If the DP-Slave transmits more diagnostic information than the DP_Master is able to process in its diagnostic buffer, the DP-Master sets the bit Diag.Ext_Diag_Overflow. If there are more diagnostic information pending at the DP-Slave than can be transmitted, the following is to note:

- it is allowed to truncate only on the block limits of the device related, identifier related or channel related diagnostic.
- if in the length field of the device related diagnostic or the identifier related diagnostic a length unequal to zero is entered, this marks a complete (not truncated) diagnostic block.

For efficiency reasons it is allowed to transmit a Diag_Data field with a fixed length. In this case the unused bytes following Ext_Diag_Data shall be filled with zero at the DP-Slave and/or the DP-Master (class 1).

9.3.2 Transfer Input and Output Data

This function permits the local user of the DP-Master to transmit output data to a DP-Slave and at the same time to request input data from this remote station. The number of input and output data which are reserved by the DP-Slave are checked against the configuration data during the start-up phase of the DP-System. If diagnostic messages or errors occur in the DP-Slave, the existence of

these messages is indicated to the DP-Master by means of a high prior response frame (interpretation of Diag_Flag).

Table 9. DDLM_Data_Exchange

Parameter Name	.req	.ind	_Upd .req	.con
Rem_Add	M			M
Outp_Data	U	U		
Status				M
Diag_Flag			M	M
Inp_Data			U	U

Rem_Add:

The parameter Rem_Add (Remote_Address) specifies the FDL address of the remote station.
Type: Unsigned8
Range: 0 to 126

Outp_Data:

This parameter contains the output data. Normally the data conveyed in this parameter are directly output to the periphery at the DP-Slave. If the Sync-Mode is activated these data are buffered in the DP-Slave and are not output to the periphery prior to renewed synchonization.
Type: Octet String
Length: Preferably 0 to 32 (extendible to 244, see section "Restrictions")

Status:

The parameter Status indicates success or failure of the function.
Possible values: OK,DS,NA,RS,RR,UE,RE

Diag_Flag:

This parameter indicates if a diagnostic exists at the DP-Slave. These diagnostic information are fetched by the DP-Master (class 1) with the function DDLM_Slave_Diag.
Type: Boolean
True: diagnostic existent
False: No diagnostic information

Inp_Data:

This parameter contains the input data of the DP-Slave.
Normally the delivered data in this parameter reflect the direct picture of the DP-Slave periphery. If the Freeze mode is activated these input data are read from an intermediate buffer. The data represent the state of DP-Slave periphery in the moment of the last Freeze control command.
Type: Octet String
Length: Preferably 0 to 32 (extendible to 244, see section "Restrictions")

9.3.3 Read Input and Output Data of a DP-Slave

By means of these functions a DP-Master (class 2) can read a picture of inputs and outputs of a DP-Slave. The condition for this is that the DP-Slave is already in the user data exchange mode.

Table 10. DDLM_RD_Inp

Parameter Name	.req	_Upd .req	.con
Rem_Add	M		M
Status			M
Inp_Data		M	C

Table 11. DDLM_RD_Outp

Parameter Name	.req	_Upd .req	.con
Rem_Add	M		M
Status			M
Outp_Data		M	C

Rem_Add:

The parameter Rem_Add (Remote_Address) specifies the FDL address of the remote station.
Type: Unsigned8
Range: 0 to 126

Inp_Data:

This parameter contains the input data of the DP-Slave.
Type: Octet String
Length: Preferably 0 to 32 (extendible to 244, see section "Restrictions")

Outp_Data:

This parameter contains the output data.
Type: Octet String
Length: Preferably 0 to 32 (extendible to 244, see section "Restrictions")

Status:

The parameter Status indicates success or failure of the function.
Possible values: OK,DS,NA,RS,UE,NR,RE

9.3.4 Send Parameter Data

By means of this function parameter data are delivered to the DP-Slaves. The parameterization of the passive stations is first done in the start-up phase of the DP system and is also possible in the user data exchange mode.
In addition to the bus wide parameter data, DP-Slave specific parameters (e. g. upper and lower limit value) can be transferred to each DP-Slave. These data are delivered from the user in the master parameter set which is buffered in the master.

Tabele 12. DDLM_Set_Prm

Parameter Name	.req	.ind	.con
Rem_Add	M		
Req_Add		M	
Prm_Data	M	M	
Status			M

Rem_Add:

The parameter Rem_Add (Remote_Address) specifies the FDL address of the remote station.
Type: Unsigned8
Range: 0 to 126

Req_Add:

This parameter contains the station address of the requester. With the help of this address the access protection in the DP-Slave is secured.
Type: Unsigned8
Range: 0 to 125

Prm_Data:

The parameter data consist of bus specific data and DP-Slave specific data.
Type: Octet String
Length: Preferably 7 to 32 (extendible to 244, see section "Restrictions")

The parameter data have the following format:

Octet 1: Station_status

	MSB						LSB
Bit-No	7	6	5	4	3	2	1	0

The individual bits have the following meaning:

Bit 7: Lock_Req
Function see table 13

Bit 6: Unlock_Req
Function see table 13

Bit 5: Sync_Req
With this bit it is indicated to the DP-Slave to operate in the Sync mode as soon as the control command is delivered by means of the DDLM_Global_Control function. If a DP-Slave does not support the Sync control command, it sets the bit Diag.Not_Supported within the diagnostic information. As a check is done during parameterization phase, errors are avoided during the user data exchange mode.

Bit 4: Freeze_Req
With this bit it is indicated to the DP-Slave to operate in the Freeze mode as soon as the control command is delivered by means of the DDLM_Global_Control function. If a DP-Slave does not support the Freeze control command, it sets the bit Diag.Not_Supported within the diagnostic information. As a check is done during parameterization phase, errors are avoided during the user data exchange mode.

Bit 3: WD_On (Watchdog on)
If this bit is set to 0 then the watchdog control is deactivated.
If the bit is set to 1 then the watchdog control is activated.

Bit 2: reserved
Bit 1: reserved
Bit 0: reserved

The term "reserved" specifies that these bits are reserved for future function extensions. If a "reserved" bit is set in a device without such function extensions the bit Diag.Not_Supported will be set.

Table 13. Specification of the bits Lock_Req and Unlock_Req

bit 7	bit 6	meaning
0	0	The paramter min T_{SDR} can be changed. All other parameters remain unchanged.
0	1	The DP-Slave will be unlocked for other masters
1	0	The DP-Slave is locked for other masters, all parameters are accepted (exception: min T_{SDR} = 0)
1	1	The DP-Slave is unlocked for other masters

Octet 2: WD_Fact_1
Range: 1 to 255
Octet 3: WD_Fact_2
Range: 1 to 255
The values entered in these two bytes represent factors for setting the watchdog control (T_{WD}). The watchdog control in a DP-Slave takes care that, if the master fails, the outputs fall in the safe state after the expiration of this time.

The time is calculated according to equation (8):

$$T_{WD} \text{ in s} = 10\text{ms} \cdot \text{WD\_Fact\_1} \cdot \text{WD\_Fact\_2} \quad (8)$$

Times can be realized between 10ms and 650s independent of the baud rate. The watchdog control is switched on or off by the bit WD_On.

Octet 4: Min. Station Delay Responder (min T_{SDR})
DP operation: Range: 0 to max T_{SDR} (see table 7)
Mixed operation: Range: 0 to 255 t_{Bit}
This is the minimum waiting time for a DP-Slave until it is allowed to send the response frames to the DP-Master. If in this octet 00H is entered, the previous value remains unchanged.

Octet 5 to 6 (unsigned16): Ident_Number
The DP-Slave accepts only parameter frames if the transmitted Ident_Number is identical with the own Ident_number. Exception: The min T_{SDR} is also allowed to be set if both bits Lock_Req and Unlock_Req are zero and the Ident_Number is not identical.

Octet 7: Group_Ident
With this octet it is possible to build groups for the function DDLM_Global_Control.
Each bit represents a group. The Group_Ident is only accepted if the Lock_Req bit is set.

Octet 8 to 32: User_Prm_Data (extendible to 244, see section "Restrictions")
The following bytes are freely allocatable for DP-Slave specific parameters, for Example diagnostic filter or regulator parameters. The meaning and the ranges of values for the User_Prm_Data are defined application specific.

Status:

The parameter Status indicates success or failure of the transfer.
Possible values: OK,DS,NA,RS,RR,UE,RE

9.3.5 Check Configuration Data

This function allows the DP-Master to deliver the configuration data to the DP-Slave for checking. They contain the range of input and output areas as well as the information about the data consistency as e. g. required for way setpoints for positioning axis.
This consistency also affects DP-Slave and DP-Master (class 1). If a DP-Master expects consistency of a DP-Slave data area, it reports this using the function DDLM_Chk_Cfg with a set consistency bit. If the DP-Slave needs consistency for a data area, it reports this using the function DDLM_Get_Cfg with a set consistency bit. In this case the consistency bit for this data area of the DP-Master shall also be set at the function DDLM_Chk_Cfg.
The DP-Slave compares its real configuration (Real_Cfg_Data) with the configuration received from the DP-Master (Cfg_Data). When verifying the configuration, the format and the length information as well the input/output areas shall be identical. The verification of the consistency bit causes only in the following cases a configuration fault (Diag.Cfg_Fault):
- The DP-Slave needs consistency for a data area and the DP-Master indicates no consistency.
- The DP-Slave can not provide consistency for a data area and the DP-Master requires consistency for this data area.

Table 14. DDLM_Chk_Cfg

Parameter Name	.req	.ind	.con
Rem_Add	M		
Req_Add		M	
Cfg_Data	M	M	
Status			M

Rem_Add:

The parameter Rem_Add (Remote_Address) specifies the FDL address of the remote station.
Type: Unsigned8
Range: 0 to 126

Req_Add:

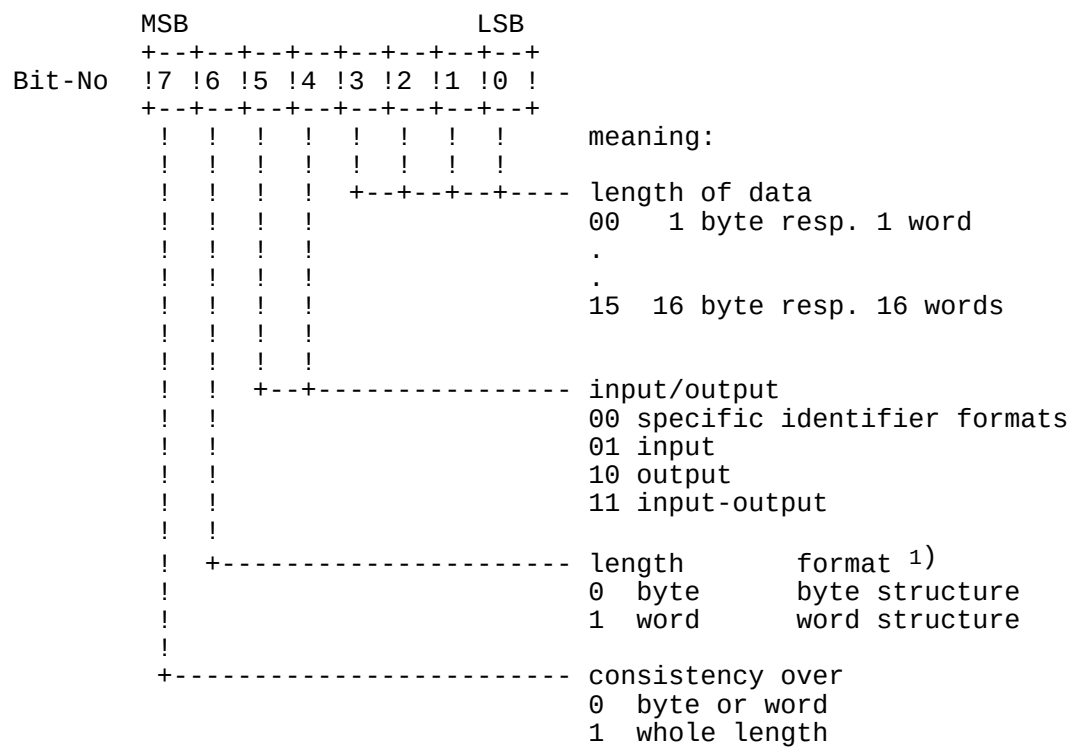
This parameter contains the station address of the requester. With the help of this address the access protection in the DP-Slave is secured.
Type: Unsigned8
Range: 0 to 125

Cfg_Data:

This parameter contains the configuration data.
Type: Octet String
Length: Preferably 1 to 32 (extendible to 244, see section "Restrictions")

In case of a modular DP-Slave one identifier byte per module should be used. Only that way provides a means for an assignment of the diagnostic information (Ext_Diag_Data) to the modules.
Input and output areas can be combined in groups and are described each by one identifier byte.

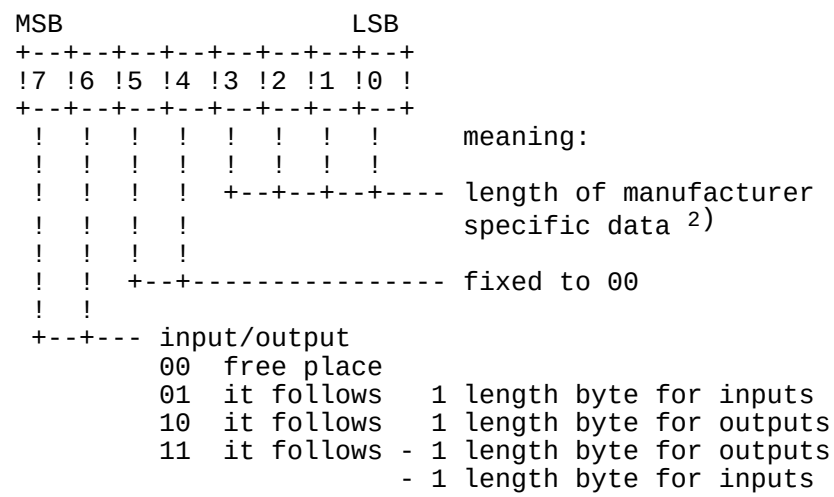
The identifier byte has the following format:



To include also extended configurations PROFIBUS-DP offers a special extension of the real identifier system to increase flexibility.
This specific identifier format makes it additionally possible to determine the number of input and output bytes associated with this identifier. In addition to this it is possible to add user specific data.

1) When transferring words, PROFIBUS-DP transfers the high byte first, followed by the low byte. If word structure is entered in the column format, the DP-Master has the possibility to swap the bytes within the word, if required by the target system.

Special identifier formats (replaces identifier byte):



---

2) The length information of the manufacturer specific data shall be interpreted as follows:

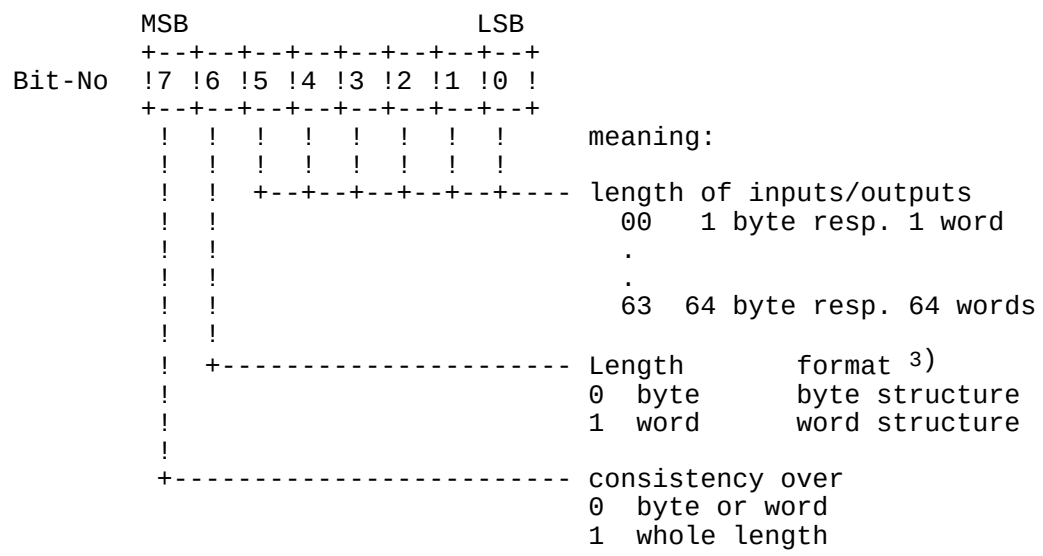
In case of DDLM_Chk_Cfg:

0	No manufacturer specific data follow; no data in Real_Cfg_Data.
1 to 14	Manufacturer specific data of specified length follow; these shall be identical with the data in Real_Cfg_Data.
15	No manufacturer specific data follow; the verification can be omitted

In case of DDLM_Get_Cfg :

0	No manufacturer specific data follow
1 to 14	Manufacturer specific data with specified length follow
15	Not allowed

The structure of the length bytes is as follows:



The following gives an Example for a **special identifier format**:

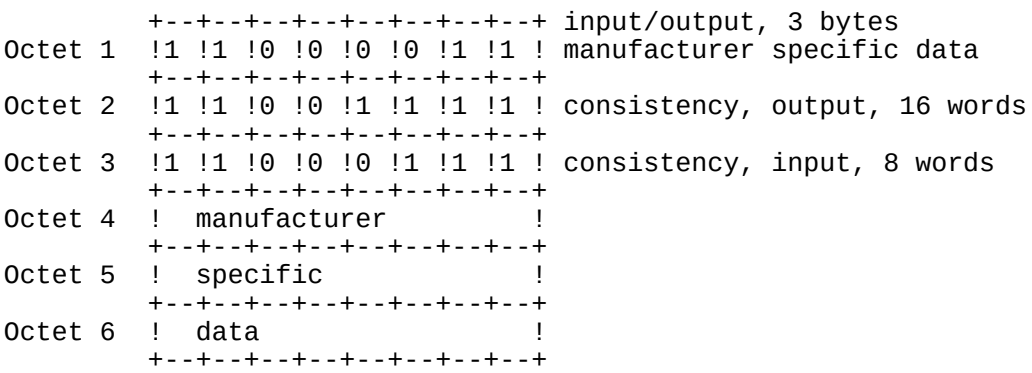


Figure 16. Example of a special identifier format

Status:

The parameter Status indicates success or failure of the transmission.
Possible values: OK,DS,NA,RS,RR,UE,RE

3) see page 736

9.3.6 Read Configuration Data
This function permits the user to read the configuration data of the DP-Slave over the communication medium. The requester gets the actual configuration (Real_Cfg_Data) which the DP-Slave has determined.

Table 15. DDLM_Get_Cfg

Parameter Name	.req	.con	_Upd .req
Rem_Add	M	M	
Status		M	
Real_Cfg_Data		C	M

Rem_Add:

The parameter Rem_Add (Remote_Address) specifies the FDL address of the remote station.
Type: Unsigned8
Range: 0 to 126

Real_Cfg_Data:

This parameter contains the configuration data. For a description of the identifiers see section "Check Configuration Data".
Type: Octet String
Length: Preferably 1 to 32 (extendible to 244, see section "Restrictions")

Status:

The parameter Status indicates success or failure of the function.
Possible values: OK,DS,NA,RS,UE,NR,RE

9.3.7 Control Commands to DP-Slave

This function permits to send special control commands to one (single) DP-Slave or to several (multicast) DP-Slaves.
For Example, this can be necessary for synchronization of the DP-Slaves. The DP-Slave accepts these control commands only from the DP-Master which has delivered parameter and configuration data. The DP-Master (class 1) uses this control commands to inform the DP-Slaves about his operation mode.

Table 16. DDLM_Global_Control

Parameter Name	.req	.ind	.con
Rem_Add	M		M
Req_Add		M	
Control_Command	M	M	
Group_Select	M	M	
Status			M

Rem_Add:

The parameter Rem_Add (Remote_Address) specifies the FDL address of the remote station.
Type: Unsigned8
Range: 0 to 126,127 (global address)

Req_Add:

This parameter contains the station address of the requester. With the help of this address access protection in the responder is secured.
Type: Unsigned8
Range: 0 to 125

Control_Command:

This parameter defines the command which shall be executed.
Type: Unsigned8
Format:

	MSB						LSB
Bit-No	7	6	5	4	3	2	1	0

The individual bits have the following meaning:

Bit 7: reserved
Bit 6: reserved

Bit 5: Sync
The output states which are transferred using the function DDLM_Data_Exchange are given out and frozen. The output data which follow are not given out as long as the next Sync control command is received.

Bit 4: Unsync
The Unsync control command cancels the Sync command.

Bit 3: Freeze
The states of the inputs are read and frozen. A further Freeze control command repeats this procedure. DP-Slaves which support the freeze mode has to ensure that already in the next data exchange cycle after the Freeze control command the last frozen values of the inputs have to be transferred.

Bit 2: Unfreeze
Freezing the inputs will be canceled.

Bit 1: Clear_Data
All outputs are cleared.

Bit 0: reserved

The term "reserved" specifies that these bits are reserved for future extensions. If a DP-Slave have this function extensions not implemented and if such a bit is set then the bit Diag.Not_Supported will be set and the DP-Slave leaves the user data exchange mode.

Table 17. Specification of the bits for Un-/Sync and Un-/Freeze

bit 2 resp. 4	bit 3 resp. 5	meaning
0	0	no function
0	1	function is activated
1	0	function is deactivated
1	1	function is deactivated

Group_Select:

This parameter determines which group(s) shall be addressed. The control command takes effect if the AND operation between the Group_Ident (delivered in the parameter data) and Group_Select results in a value different from zero. If the parameter Group_Select is zero all DP-Slaves are addressed.
Type: Unsigned8

Status:

The parameter Status indicates only whether transmission of the request frame was successful or not.
Possible values: OK,DS,NO,IV

9.3.8 Change Station Address of a DP-Slave

This function permits a DP-Master (class 2) to change the address of a DP-Slave. If the DP-Slave has not the possibility to store (EEPROM,FLASH) or if the address setting is realized with a switch, this function is refused with RS error message. At the same time the Ident_Number is transmitted with this function. The station address will be changed if the local and the transferred Ident_Number correspond.

Table 18. DDLM_Set_Slave_Add

Parameter Name	.req	.ind	.con
Rem_Add	M		M
New_Slave_Add	M	M	
Ident_Number	M	M	
No_Add_Chg	M	M	
Rem_Slave_Data	U	C	
Status			M

Rem_Add:

The parameter Rem_Add (Remote_Address) specifies the previous FDL address of the remote station.
Type: Unsigned8
Range: 0 to 125;126 (default address)

New_Slave_Add:

This parameter contains the new DP-Slave address which is to be set.
Type: Unsigned8
Range: 0 to 125

Ident_Number:

This number specifies the station type (see section "Manufacturer Identifier").
Type: Unsigned16

No_Add_Chg:

This parameter specifies whether it is allowed to change the DP-Slave address again at a later date. If this is not allowed then it is only possible to change the address with this function after the initial reset. After the initial reset the DP-Slave takes the default address 126.
Type: Boolean
True: change of address is only possible after the initial reset
False: change of address is also possible later

Rem_Slave_Data:

With this parameter it is possible to deliver user specific data. The data are stored in the DP-Slave if there exists a possibility (EEPROM, FLASH).
Type: Octet String
Length: Preferably 0 to 28 (extendible to 240, see section "Restrictions")

Status:

The parameter Status indicates if the request frame was sent successfully and was accepted by the DP-Slave.
The acknowledgement does not contain whether the new values were accepted by the DP-Slave.
The master shall check the correct execution of the function by using DDLM_Slave_Diag.req with the new DP-Slave address.
Possible values: OK,DS,NA,RS,RR,UE,RE

9.4 DP-Master - DP-Master Functions

9.4.1 Read Master Diagnostic Information

With this function it is possible to request both the diagnostic information of its assigned DP-Slaves deposited in the DP-Master (class 1) and its own status. The diagnostic information are structured in a system wide diagnostic information collection and in a station related single diagnostic. Which diagnostic is requested depends on the request parameters of the function.

Table 19. DDLM_Get_Master_Diag

Parameter Name	.req	.ind	.res	.con
Rem_Add	M			
Identifier	M	M		
Status			M	M
Diagnostic_Data			C	C
Diag_Data			S	S
System_Diagnostic			S	S
Data_Transfer_List			S	S
Master_Status			S	S

Rem_Add:

The parameter Rem_Add (Remote_Address) specifies the FDL address of the DP-Masters from which the diagnostic information are requested.
Type: Unsigned8
Range: 0 to 125

Identifier:

This parameter specifies the type of the requested diagnostic information or the FDL address of a DP-Slave.
Type: Unsigned8
Range:

0-125	≥	Diag_Data of the DP-Slave
126	≥	System_Diagnostic
127	≥	Master_Status
128	≥	Data_Transfer_List
129-255	≥	reserved

Status:

The parameter Status indicates success or failure of the function.
Possible values: OK,DS,NA,RS,RR,UE,RE,TO,FE,NE,IP,AD,EA,LE

```
Diagnostic_Data:

Different diagnostic information are transmitted depending on Identifier.

Diag_Data (Identifier = 0 to 125):
This parameter is described in section "Read DP-Slave Diagnostic Information".

System_Diagnosic (Identifier = 126):
Type: Octet String
Length: 16

Octet 1:
  bit 0 = 1 station number 0 station has reported diagnostic
  bit 0 = 0 station number 0 station has reported no diagnostic

  bit 1 = 1 station number 1 station has reported diagnostic
  bit 1 = 0 station number 1 station has reported no diagnostic

etc.

Octet 16:
  .
  .

  bit 5 = 1 station number 125 station has reported diagnostic
  bit 5 = 0 station number 125 station has reported no diagnostic

  bit 6, bit 7 = 0 (not used)


Master_Status (Identifier = 127):
Type: Octet String
Length: 16

Octet 1: USIF_State (see section "Set Operation Mode")
         40H    = Stop
         80H    = Clear
         C0H    = Operate

Octet 2 and Octet 3 (unsigned16): Ident_Number

Octet 4: Hardware Release (DDLM/User-Interface)

Octet 5: Firmware Release (DDLM/User-Interface)

Octet 6: Hardware Release (User)

Octet 7: Firmware Release (User)

Octet 8 to 16:
         reserved
```

Data_Transfer_List (Identifier = 128):
Type: Octet String
Length: 16
Meaning:
The user data exchange mode between the DP-Master (class 1) and its assigned DP-Slaves is supervised on the master side with Data_Control_Time. By this means it is checked whether within a projectable time an user data transfer to the associated DP-Slaves was executed at least once. The Data_Transfer_List is updated at least once within Data_Control_Time.

Octet 1:
  bit 0 = 1 user data exchange executed with station number 0
  bit 0 = 0 no user data exchange with station number 0

  bit 1 = 1 user data exchange executed with station number 1
  bit 1 = 0 no user data exchange with station number 1

etc.

Octet 16:
  .
  .

  bit 5 = 1 user data exchange executed with station number 125
  bit 5 = 0 no user data exchange with station number 125

  bit 6, bit 7 = 0 (not used)

9.4.2 Up-/Download

These functions permit a DP-Master (class 2) to transfer or to read a data area to or from another DP-Master (class 1). An access protection can be guaranteed with the functions DDLM_Start_Seq and DDLM_End_Seq.

Description of the interactions of the Upload and Download functions:

- DDLM_Download

The beginning of the Download function can be signalized with the function DDLM_Start_Seq. The actual data set is transmitted in blocks with the function DDLM_Download in the area of the DP-Master marked Area_Code .
After the complete transmission and possible activation, the process shall be terminated with the function DDLM_End_Seq, if the function DDLM_Download was started with the function DDLM_Start_Seq.

- DDLM_Upload

The Upload is executed in the same way as the Download. It is possible to transfer single data areas without the functions DDLM_Start_Seq and DDLM_End_Seq. Consistency of the transferred data is not guaranteed in this case.

Table 20. DDLM_Start_Seq

Parameter Name	.req	.ind	.res	.con
Rem_Add	M			
Req_Add		M		
Area_Code	M	M		
Timeout	M	M		
Status			M	M
Max_Len_Data_Unit			C	C

Table 21. DDLM_Download

Parameter Name	.req	.ind	.res	.con
Rem_Add	M			
Req_Add		M		
Area_Code	M	M		
Add_Offset	M	M		
Data	M	M		
Status			M	M

Table 22. DDLM_Upload

Parameter Name	.req	.ind	.res	.con
Rem_Add	M			
Req_Add		M		
Area_Code	M	M		
Add_Offset	M	M		
Data_Len	M	M		
Status			M	M
Data			C	C

Table 23. DDLM_End_Seq

Parameter Name	.req	.ind	.res	.con
Rem_Add	M			
Req_Add		M		
Status			M	M

Rem_Add:

The parameter Rem_Add (Remote_Address) specifies the FDL address of the remote station.
Type: Unsigned8
Range: 0 to 125

Req_Add:

This parameter contains the station address of the requester. With the help of this address access protection in the responder is secured.
Type: Unsigned8
Range: 0 to 125

Area_Code:

The parameter Area_Code identifies the area which shall be loaded or read. If the parameter Area_Code of the function DDLM_Start_Seq is set to a value of 255, local access protection for a determined Area_Code is not guaranteed for the functions following. The local access protection ensures that the user cannot access the partially loaded data during the sequence.
Type: Unsigned8
Range:

0 to 125	DP-Slave parameter set (description see section "Coding of Parameter Set of DP-Slave")
126	reserved
127	Bus parameter set (description see section "Coding of the Bus Parameter Set")
128	reserved
129	statistic counters (description see section "Coding of Statistic Counters")
130 to 135	reserved for diagnostic message filter
136 to 139	reserved for the transfer of the master parameter set
140 to 254	reserved
255	DDLM_Start_Seq: No local access protection

At the user in the DP-Master (class 1) an intermediate buffer shall be made available which is only reserved for storing the bus parameter set. The Download function transfers the bus parameter set into the intermediate buffer. With the functions DDLM_Act_Param or DDLM_Act_Para_Brct the bus parameter set is taken by the User-Interface with the help of the function Load_Bus_Par from the intermediate buffer into the active buffer. Then the bus parameter set will be activated and retained unchanged in the intermediate buffer. The Upload function always reads out the active buffer.

Timeout:

This parameter defines the control time between two successive Upload/Download functions. If this time expires the access protection is deactivated. The previously transferred data are invalid.
Type: Unsigned16
Time base: 1ms

Add_Offset:

This parameter identifies the offset in bytes to the begin of the area which is specified by the Area_Code.
Type: Unsigned16

Data:

This parameter contains the area data to be transferred.
Type: Octet String
Length: 1 to 240

Data_Len:

The parameter defines the length of the requested data.
Type: Unsigned8
Range: 1 to 240

Status:

The parameter Status indicates success or failure of the function.
Possible values:

DDLM_Start_Seq:	OK,DS,NA,RS,RR,UE,TO,FE,RE,NE,AD,IP,NI, SE,SC,EA,LE,RE
DDLM_Download:	OK,DS,NA,RS,RR,UE,TO,FE,RE,NE,AD,EA,LE,SC,NI,NC
DDLM_Upload:	OK,DS,NA,RS,RR,UE,TO,FE,RE,NE,EA,LE,NI,SC,AD
DDLM_End_Seq:	OK,DS,NA,RS,RR,UE,TO,FE,RE,NI,SE,NE,AD,EA,LE,NC

Max_Len_Data_Unit:

The parameter Max_Len_Data_Unit defines the maximum possible length of the parameter Data in the following Upload/Download frames.
Type: Unsigned8
Range: 1 to 240

9.4.3 Activate Parameter Set (unconfirmed)

After the Download of a parameter set this can be accepted and activated with the function DDLM_Act_Para_Brct.
This function can be sent simultaneously to one (single) or many (multicast) DP-Master (class 1) in dependence of the parameter Rem_Add.
In the operation modes Clear and Operate of the DP-Master (class 1) User-Interface, the activation of the bus parameter set with changed FDL parameters Baud_Rate or FDL_Add will not take place.
This function is not acknowledged by the receiver (reason: a possible change of baud rate must take effect at all addressed DP-Masters nearly at the same time).

Table 24. DDLM_Act_Para_Brct

Parameter Name	.req	.ind	.con
Rem_Add	M		
Area_Code	M	M	
Status			M

Rem_Add:

The parameter Rem_Add (Remote_Address) specifies the FDL address of the remote station.
Type: Unsigned8
Range: 0 to 125,127

Area_Code:

The parameter Area_Code identifies the area which shall be activated.
Type: Unsigned8
Range:

0 to 126	not allowed
127	bus parameter set
128 to 129	reserved
130 to 135	reserved for diagnostic message filter
136 to 139	reserved for the master parameter set
140 to 254	reserved
255	not allowed

In the presence of several DP-Masters (class 2) or FMS stations, it shall be guaranteed that the old and the new bus parameter sets are compatible.

Status:

The parameter Status indicates only whether the transmission of the request frame was successful or not.
Possible values: OK,DS

9.4.4 Activate/Deactivate Parameter Set

With the help of this function it is possible at the DP-Master (class 1)

- to activate or to deactivate a parameterized DP-Slave or
- to change the operation mode of the User-Interface or
- to accept and activate the bus parameter set.

If the loaded bus parameter set contains a changed baud rate or station address then the new bus parameter set shall be activated with the function DDLM_Act_Para_Brct.

Table 25. DDLM_Act_Param

Parameter Name	.req	.ind	.res	.con
Rem_Add	M			
Area_Code	M	M		
Activate	M	M		
Status			M	M

Rem_Add:

The parameter Rem_Add (Remote_Address) defines the FDL address of the DP-Master at which the function DDLM_Act_Param shall be executed.
Type: Unsigned8
Range: 0 to 125

Area_Code:

The parameter Area_Code identifies the area which shall be activated/deactivated with the exception of the operation mode.
Type: Unsigned8
Range:

0 to 125	DP-Slave parameter set The Active Flag in the DP-Slave parameter set of the DP-Master (class 1) is influenced accordingly. Thereupon, the DP-Slave concerned takes part in the cyclic user data exchange mode or is removed from this mode and no longer addressed.
126	reserved
127	bus parameter set
128	operation mode (description see section "Set Operation Mode")
129	reserved
130 to 135	reserved for diagnostic messages filter
136 to 139	reserved for activation of the master parameter set
140 to 255	reserved

Activate:

The meaning and the possible values of this parameter depend on the Area_Code.
Type: Unsigned8
Area_Code: 0 to 125 (DP-Slave parameter set)
Activate: 80H
Deactivate: 00H

Area_Code: 127 (bus parameter set)
Activate: 255

Area_Code: 128 (operation mode)
Stop: 40H
Clear: 80H
Operate: C0H

Status:

The parameter Status indicates success or failure of the function.
Possible values: OK,DS,NA,RS,RR,UE,TO,FE,RE,NE,AD,IP,SC,NI,DI,EA,LE

9.5 Local Functions at DP-Slave

9.5.1 General

On the interface to the User-Interface the DDLM of the DP-Slave makes available the functions for local interactions between the User-Interface and the DDLM.
The DDLM maps the DDLM functions onto the confirmed services of FDL and FMA1/2.
The DDLM performs the handling of the local service confirmation independent of FDL and FMA1/2. If a local error occurs during the service execution in the FDL or in FMA1/2 then a DDLM_Fault.ind is generated from DDLM.

9.5.2 DDLM_Slave_Init

With this function the local FDL and the local FMA1/2 is initialized at the DP-Slave for data communication and the local station address is delivered.

This function is mapped from the DDLM onto the following services of FMA1/2:

- Reset FMA1/2
- Set Value FMA1/2
- RSAP Activate FMA1/2

Table 26. DDLM_Slave_Init

Parameter Name	.req
Station_Address	M
Baud_Rate	U
Medium_red	U

In implementations where the layer 2 bus parameters are locally known in layer 2, the parameters Baud_Rate and Medium_red may be omitted.

Station_Address:

The parameter Station_Address specifies the local FDL address.

Baud_Rate:

The parameter Baud_Rate specifies the baud rate locally set.

Medium_red:

The parameter Medium_red specifies whether the system does or does not use a redundant transmission technique.

9.5.3 DDLM_Set_minTsdr

With this function the operation parameter minTsdr is delivered to the local FDL at the DP-Slave.
This function is mapped from the DDLM onto the following FMA1/2 service:

- Set Value FMA1/2

Table 27. DDLM_Set_minTsdr

Parameter Name	.req
minTsdr	M

minTsdr:

The parameter minTsdr specifies the value for the minimum station delay time (min T_{SDR}) at the responder. If this value is zero then no change will take effect.

9.5.4 DDLM_Enter

With this function the local FDL and the local FMA1/2 of the DP-Slave are initialized for the user data exchange mode of the DDLM. At the same time the assigned SAPs for the output data and the function DDLM_Global_Control are activated and parameterized with a layer 2 access protection.
This function is mapped from the DDLM onto the following services of FMA1/2:

- RSAP Activate FMA1/2
- SAP Activate FMA1/2

Table 28. DDLM_Enter

Parameter Name	.req
Master_Add	M

Master_Add

The parameter Master_Add specifies the FDL address of the master which gets access for direct writing of the outputs and for requesting the function DDLM_Global_Control.

9.5.5 DDLM_Leave

With this function the local FDL and the local FMA1/2 of the DP-Slave are reconfigured and the user data exchange mode of the DDLM is terminated.
At the same time layer 2 access protection, the assigned SAPs for the output data and the function DDLM_Global_Control in the layer 2 are deactivated. This function has no input parameters.

This function is mapped from the DDLM onto the following FMA1/2 service:

- SAP Deactivate FMA1/2

9.5.6 DDLM_Fault

With this function the DDLM of the DP-Slave indicates an error to the User-Interface during the execution of a local FDL or FMA1/2 service. This function has no input and output parameters.

This function will be transferred by the DDLM to the User-Interface after the occurrence of one of the following events:

- FMA1/2_RESET.con (NO/IV)
- FMA1/2_SET_VALUE.con (NO/IV)
- FMA1/2_RSAP_ACTIVATE.con (NO/IV)
- FMA1/2_SAP_ACTIVATE.con (NO/IV)
- FMA1/2_SAP_DEACTIVATE.con (NO/IV)
- FDL_REPLY_UPDATE.con (LS/LR/IV)
- FDL_DATA_REPLY.ind (DSAP inadmissible)
- FDL_DATA.ind (DSAP inadmissible)
- unknown FDL or FMA1/2 primitive

9.6 Local Functions at DP-Master

9.6.1 General

At the interface to the User-Interface the DDLM of the DP-Master provides functions for local interactions between the User-Interface and the DDLM.
The DDLM maps the DDLM functions to the confirmed services of FDL and FMA1/2. The treatment of local service confirmations will be done by the DDLM. If a local error occurs in the FDL or in the FMA1/2 during execution of a local service, the DDLM will produce a DDLM_Fault.ind.

9.6.2 DDLM_Master_Init

With this function the User-Interface of the DP-Master prepares the DDLM (Requester) for communication with DP-Slaves.

This function will be mapped by the DDLM on the following service of FMA1/2:

- SAP Activate FMA1/2

This function has no parameters.

9.6.3 DDLM_Responder_Init

With this function the User-Interface of the DP-Master (class 1) prepares the DDLM (Responder) for communication with a DP-Master (class 2).

This function will be mapped by the DDLM on the following services of FMA1/2:

- RSAP Activate FMA1/2
- SAP Activate FMA1/2

Table 29. DDLM_Responder_Init

Parameter Name	.req	.con
Poll_Timeout	M	

Poll-Timeout:

The parameter Poll_Timeout specifies the maximum period of time during which the response has to be picked up by the requester.

9.6.4 DDLM_Requester_Init

With this function the User-Interface of the DP-Master (class 2) prepares the DDLM (Requester) for communication with a DP-Master (class 1).

This function will be mapped by the DDLM on the following service of the FMA1/2:

- SAP Activate FMA1/2

Table 30. DDLM_Requester_Init

Parameter Name	.req	.con
Poll_Timeout	M	

Poll-Timeout:

The parameter Poll_Timeout specifies the maximum period of time in which the response has to be received from the responder.

9.6.5 DDLM_Reset

With this function the local FDL and FMA1/2 will be reset.

This function will be mapped by the DDLM on the following service of the FMA1/2:

- Reset FMA1/2

This function has no parameters.

9.6.6 DDLM_Set_Bus_Par

With this function the layer 2 parameters from the bus parameter set will be passed to the FDL.

This function will be mapped by the DDLM on the following service of the FMA1/2:

- Set Value FMA1/2

Table 31. DDLM_Set_Bus_Par

Parameter Name	.req	.con
Bus_Para	M	
Status		M

Bus_Para:

Description see section "Data Interface".

Status:

The parameter Status indicates success or failure of the function.
Possible values: OK,NO,IV

9.6.7 DDLM_Set_Value

With this function the FDL parameters can be passed to FMA1/2.

This function will be mapped by the DDLM on the following service of the FMA1/2:

- Set Value FMA1/2

Table 32. DDLM_Set_Value

Parameter Name	.req	.con
Variable	M	
Value	M	
Status		M

Variable:

This parameter selects the FDL parameter which has to be changed.

Value:

This parameter contains the new value for the FDL parameter.

Status:

The parameter Status indicates success or failure of the function.
Possible values: OK,NO,IV

9.6.8 DDLM_Read_Value

With this function it is possible to read the FDL Variables.
This function will be mapped by the DDLM on the following service of the FMA1/2:

- Read Value FMA1/2

Table 33. DDLM_Read_Value

Parameter Name	.req	.con
Variable	M	
Status		M
Value		C

Variable:

The parameter Variable selects the FDL Variable which has to be read.

Value:

This parameter contains the actual value of the read FDL parameter.

Status:

The parameter Status indicates success or failure of the function.
Possible values: OK,NO,IV

9.6.9 DDLM_Delete_SC

With this function the statistic counters will be cleared (see section "Statistic Counters").

This function will be mapped by the DDLM on the following service of the FMA1/2:

- Set Value FMA1/2

Table 34. DDLM_Delete_SC

Parameter Name	.req	.con
Address	M	
Status		M

Address:

The statistic counters of the station which is addressed by the parameter Address will be cleared.

Status:

The parameter Status indicates success or failure of the function.
Possible values: OK,NO,IV

9.6.10 DDLM_Fault

With this function the DDLM indicates an error during execution of a local FDL or FMA1/2 service to the User-Interface.
This function has no input and output parameters.

This function will be passed by the DDLM to the User-Interface after the occurrence of one of the following events:

- FMA1/2_RESET.con (NO/IV)
- FMA1/2_SET_VALUE.con (NO/IV)
- FMA1/2_RSAP_ACTIVATE.con (NO/IV)
- FMA1/2_SAP_ACTIVATE.con (NO/IV)
- FMA1/2_SAP_DEACTIVATE.con (NO/IV)
- FDL_REPLY_UPDATE.con (LS/LR/IV)
- FDL_DATA_REPLY.ind (DSAP inadmissible)
- FDL_DATA.ind (DSAP inadmissible)
- unknown FDL or FMA1/2 Primitive

9.6.11 DDLM_Event

With this function the DDLM indicates an event notification of the layer 2 to the User.

Table 35. DDLM_Event

Parameter Name	.ind
Event	M
Add_Info	C

Event, Add_Info:

These parameters can take the values which are allowed for the Event FMA1/2 service (e. g. Out_Of_Ring, GAP_event).

10 Interface between User-Interface and User

10.1 DP-Master (class 1)

The interface between the User-Interface and the User at the DP-Master (class 1) is defined as a data and service interface. At this interface all interactions for the communication between the DP-Master (class 1) and the DP-Slaves will be handled. This interface will be described in detail as follows.

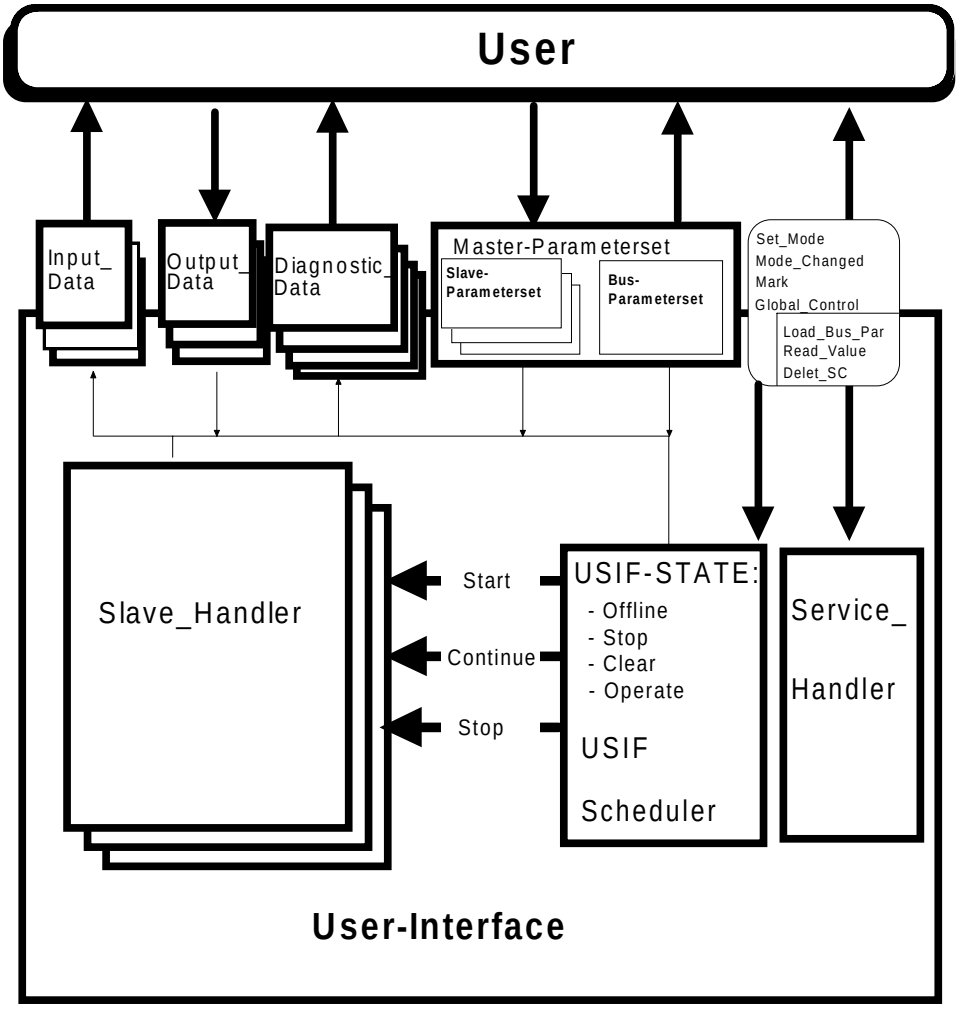


Figure 17. Interface between User and User-Interface at the DP-Master (class 1)

10.1.1 Data Interface

a) Bus Parameter Set:

The current bus parameter set is located in this area.
The bus parameter set consists of:

- FDL operational parameter

- Error_Action_Flag:

This bit will be evaluated in the operation mode Operate and Clear. If data transfer to at least one activated DP-Slave was impossible during Data_Control_Time and if the Error_Action_Flag is set, the operation mode will change from Operate to Clear. If the Error_Action_Flag is not set, the User-Interface will remain in the operation mode Operate in the case of an error.
If the operation mode of the User-Interface shall be changed from Clear to Operate, this will only be done if in the Data_Control_Time all DP-Slaves where in the user data exchange mode if the bit Error_Action_Flag was set.

- Min_Slave_Interval:

This parameter specifies the smallest allowed period of time between two slave poll cycles. This ensures that the sequence of function requests from the DP-Master can be handled by the DP-Slave. This period of time will be complied by the DP-Master (class 1) for every master-slave function with exception of the function Global_Control. For the function Global_Control the user is responsible for compliance with Min_Slave_Interval.

- Poll_Timeout:

In the case of master-master communication this parameter specifies the maximum amount of time it may take the requester to fetch the response.

- Data_Control_Time

This parameter specifies the period of time during which the Data_Transfer_List (see section "Control Intervals") will be updated at least once and will be made available to the user.

The User can only read the current bus parameter set. The User can pass a new bus parameter set to the User-Interface by means of the function Load_Bus_Par. After that, the User-Interface transfers the new bus parameter set to the current bus parameter set and loads the changed FDL operational parameters into FDL control.

b) DP-Slave parameter sets:

The User-Interface needs a DP-Slave parameter set for every DP-Slave. This parameter set contains the control bits Active and New_Prm.

- Active-Flag:
  If the Active-Flag has the value False, the DP-Master stops the DP-Slave handling to the concerned DP-Slave. In this state it is allowed to change the DP-Slave parameter set, for Example by a DP-Master (class 2) with the function DDLM_Download.
  If the Active-Flag has the value True, the DP-Slave handling will be done by the DP-Master (class 1) to the concerned DP-Slave. Changing the DP-Slave parameter set in this state is reduced possible (see also New_Prm-Flag).

- New_Prm-Flag
  If this flag has the value False, user data will be transferred to this DP-Slave in the user data exchange mode. If the flag is True, for one data cycle parameter data will be transferred instead of user data. The consistency of the parameter data has to be guaranteed by local measures. After transmitting this data successfully the DP-Master (class 1) sets the

Flag New_Prm to False and the user data exchange mode to the DP-Slave will be continued.

In case of a DDLM_Act_Param.ind the user has to clear or to set the Active-Flag in accordance with the parameter Activate in the corresponding DP-Slave parameter set. A new DP-Slave parameter set which contains the Active-Flag can only be loaded with a cleared Active-Flag. For this it is necessary that the User-Interface is in the operation mode Clear or Operate. In the operation mode Stop DP-Slave parameter sets can be loaded regardless of the state of the Active-Flag.

The data sets Prm_Data and Cfg_Data will be transferred directly from the DP-Slave parameter set to the DP-Slave during parametrization and configuration.

The Add_Tab contains the relation between the in- and output data interface of the User-Interface and the in- and output data regions of the DP-Slaves.

c) Diagnostic_Data:

In this area the diagnostic information of the DP-Slaves, the system diagnostic, the Data_Transfer_List and the Master_Status will be stored by the User-Interface.

d) Input, Output Data:

In this area the input data of the DP-Slaves and the output data of the user will be stored. The length of this area is manufacturer specific. The arrangement of the in- and output data depends on how the user designs his DP-system. This arrangement is given in the Add_Tab region of the DP-Slave parameter set.

10.1.2 Service Interface

The shaping as a service interface gives the user the possibility to call acyclic functions asynchronously to the cyclic operation of the User-Interface. They are subdivided in local and remote functions. The local functions are handled by the Scheduler or the Service-Handler. The remote functions are handled by the Scheduler. The User-Interface provides no additional error handling for these functions. At this interface the service calls shall be called sequentially. A parallel handling is only allowed in the case that a Mark.req was passed to the interface and a Global_Control.req is made.

10.1.2.1 Set Operation Mode

By means of this function the user can set the operation mode (USIF_State) of the User-Interface. The user can read the given operation mode of the User-Interface with the function DDLM_Get_Master_Diag. With the function DDLM_Download the operation mode can also be changed by the DP-Master (class 2).

Table 36. Set_Mode (local)

Parameter Name	.req	.con
USIF_State	M	
Status		M

USIF_State:

The User-Interface provides four operation modes for the user:

- Offline:
  Communication to all DP participants (DP-Master and DP-Slave) is stopped. Local FDL control will be taken out of the token ring. The User-Interface waits for a signal to start.

- Stop:
  The bus parameter set is loaded in the FDL. The FDL control is active. The responder functions for communication with the DP-Master (class 2) are processed. DP-Slaves are not polled.

- Clear:
  In this operation mode the DP-Master (class 1) tries to parameterize, check configuration and performs user data exchange with its associated DP-Slaves. The inputs of the DP-Slaves are read and passed to the user at the in- and output data interface. The outputs from the User will be ignored; data equal to zero will be transferred to the DP-Slaves.

- Operate:
  The DP-Master (class 1) is in the user data exchange mode with the dedicated DP-Slaves. The inputs from the DP-Slaves are transferred to the user and the outputs from the user are passed to the DP-Slaves. If the User-Interface leaves the operation mode Operate, the outputs of all DP-Slaves will be cleared by means of the function Global_Control (Control_Command = Clear_Data, Group_Select = 0).

Status:

The parameter Status indicates success or failure of the function.
Possible values: OK,NO,IV

10.1.2.2 Message upon Change of Operation Mode

With this function the User-Interface indicates the change of its operation mode. This indication will not occur if the user has initiated the change of the operation mode through the function Set_Mode. If the Error_Action_Flag is set the User-Interface remains in the operation mode Operate as long as the user data exchange to the activated DP-Slaves takes place successfully. The User-Interface changes the operation mode to the state Offline if a serious error happens at the local interface. This will occur independent of the Error_Action_Flag.

Table 37. Mode_Changed (local)

Parameter Name	.ind
USIF_State	M

USIF_State:
See function Set_Mode in section "Set Operation Mode".

10.1.2.3 Load Bus Parameter Set

With this function the user can load a new bus parameter set. The User-Interface transfers the new loaded bus parameter set in the current one and passes the changed FDL operational parameters to FDL control. In the operation mode Clear and Operate of the User-Interface it is not allowed to load a bus parameter set with changed FDL parameters Baud_rate or FDL_Add.

Table 38. Load_Bus_Par (local)

Parameter Name	.req	.con
Bus_Para	M	
Status		M

Bus_Para:
This parameter contains the bus parameter set which shall be loaded. The structure is similar to the current bus parameter set (see section "Coding of the Bus Parameter Set").

Status:

The parameter Status indicates success or failure of the function.
Possible values: OK,NO,IV

10.1.2.4 Synchronize Data Transfer

By means of this function the user can operate synchronously with the User-Interface. After the user has passed this function to the User-Interface, the user will receive an acknowledgement from the User-Interface after all activated DP-Salves where polled at least once.

Table 39. Mark (local)

Parameter Name	.req	.con
Status		M
Dia		C

Dia:

This parameter indicates that at least one activated DP-Slaves was not in the user data exchange mode.
Possible Values: True, False

Status:

The parameter Status indicates success or failure of the function. The value NO will be signalled if the User-Interface is in the operation mode Offline or Stop, or a Mark.req is still active.
Possible Values: OK,NO

10.1.2.5 Control Commands to DP-Slave

This function makes it possible to send the control commands Sync and Freeze to one (single) or several (multicast) DP-Slaves. This can be necessary to synchronize DP-Slaves.

Table 40. Global_Control (remote)

Parameter Name	.req	.con
Rem_Add	M	M
Control_Command	M	
Group_Select	M	
Status		M

Rem_Add:

The parameter Rem_Add (Remote_Address) fixes the FDL address of the remote participant.
Range: 0 to 126,127 (global address)

Control_Command:

This parameter specifies the command or the commands which has to be performed. The DP-Master (class 1) sets the flag Clear_Data according to his operation mode.
Possible commands:
- Sync
- Unsync
- Freeze
- Unfreeze

Group_Select:

This parameter determines which group or groups has to be addressed. The command becomes effective when the logical AND between the Group_Ident (will be sent at parametrization) and the Group_Select has a result not equal to zero. All DP-Slaves will be addressed if the parameter Group_Select has the value zero.

Status:

The parameter Status indicates only whether the request telegram could be sent or not. The case NO indicates that the Control_Command is not allowed.
Possible values: OK,DS,NO

10.1.2.6 Read Statistic Counters

This function makes it possible to read the statistic counters.

Table 41. Read_Value (local)

Parameter Name	.req	.con
Variable	M	
Status Value		M C

Variable:

The parameter Variable selects the FDL Variable whose current value has to be read.

Value:

This parameter contains the read value of the FDL parameter.

Status:

The parameter Status indicates success or failure of the function.
Possible values: OK,NO,IV

10.1.2.7 Clear Statistic Counter

With this function the statistic counters can be cleared.

Table 42. Delete_SC (local)

Parameter Name	.req	.con
Address	M	
Status		M

Address:

This parameter specifies the FDL address of the statistic counters that have to be cleared.
Allowed values:
0 to 126: FDL address
All: All statistic counters

Status:

The parameter Status indicates success or failure of the function.
Possible values: OK,NO,IV

10.1.3 Server Behaviour at DP-Master (Class 1)

At the DP-Master (class 1) the functions for the master-master communication have to be passed by the user to the data and/or service interface of the User-Interface.

a) DDLM_Get_Master_Diag: data interface: Diagnostic_Data

Identifier (0 to 125): Diag_Data from the corresponding DP-Slave (read)
Identifier (126): System_Diagnostic (read)
Identifier (127): Master_Status (read)
Identifier (128): Data_Transfer_List (read)

b) DDLM_Start_Seq: no mapping onto the User-Interface

c) DDLM_End_Seq: no mapping onto the User-Interface

```
d) DDLM_Upload:          data, service interface

  Area_Code (0 to 125): DP-Slave parameter set (read)
                        (data interface: master parameter set)

  Area_Code (127):   bus parameter set (read)
                        (data interface: master parameter set)

  Area_Code (129):   Statistic counter (read)
                        (Service: Read_Value)

  Area_Code (255):   no mapping onto the User-Interface


e) DDLM_Download:        data, service interface

  Area_Code (0 to 125): DP-Slave parameter set (write)
                        (data interface: master parameter set)

  Area_Code (127):   bus parameter set in the temporary memory of
                     the user (write)
                        (no mapping onto the User-Interface)

  Area_Code (129):   Statistic counter (delete)
                        (Service: Delete_SC)

  Area_Code (255):   no mapping onto the User-Interface


f) DDLM_Act_Para_Brct:  service interface

  Area_Code (127):   bus parameter set (load and activate)
                        (Service: Load_Bus_Par)


g) DDLM_Act_Param:       data, service interface

  Area_Code (0 to 125): DP-Slave parameter set (de-, activate)
                        (data interface: master parameter set)

  Area_Code (127):   bus parameter set (load and activate)
                        (Service: Load_Bus_Par)

  Area_Code (128):   Operation mode (set)
                        (Service: Set_Mode)
```

10.2 DP-Master (class 2)

At the DP-Master (class 2) the User-Interface is not existent. The user of the DP-Master (class 2) mapps his functions directly onto the DDLM-Interface.

10.3 DP-Slave

At the DP-Slave the interface between User and User-Interface is created as a data interface. This interface is described in section "Communication Model of the DP-Slave".

11 Coding

11.1 Coding Rules for Additional Information

11.1.1 General

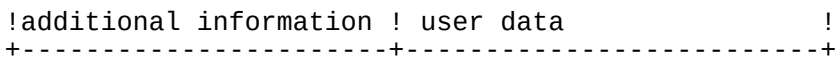
In the case of Master-Master communication the coding information of the different functions is transferred in the layer 2 Data_Unit. The coding instructions for this additional information are optimized for short messages, taking into account the special fieldbus sector requirements and security of the transferred data.
For communication with DP-Slaves additional information is not used. The distinction of the functions will be made, without exception, by means of different Service Access Points.

11.1.2 Coding Rules

The structure and the use of the additional information depends on the used DP-Master function. In this case the additional information will be part of the Data_Unit.

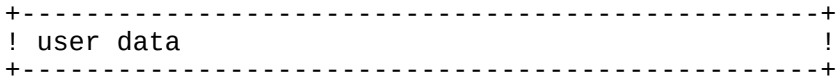
The structure of a Data_Unit is given through the insertion of additional information or through the kind of function itself.

Structure of a Data_Unit (L_sdu) with additional information:



The additional information contains no information about the length, but is given through the used function. The sequence of the parameters in the Data_Unit depends on the sequence of the parameters in the function call. After the additional information follows the user data depending on the used function.

Structure of a Data_Unit (L_sdu) without additional information:

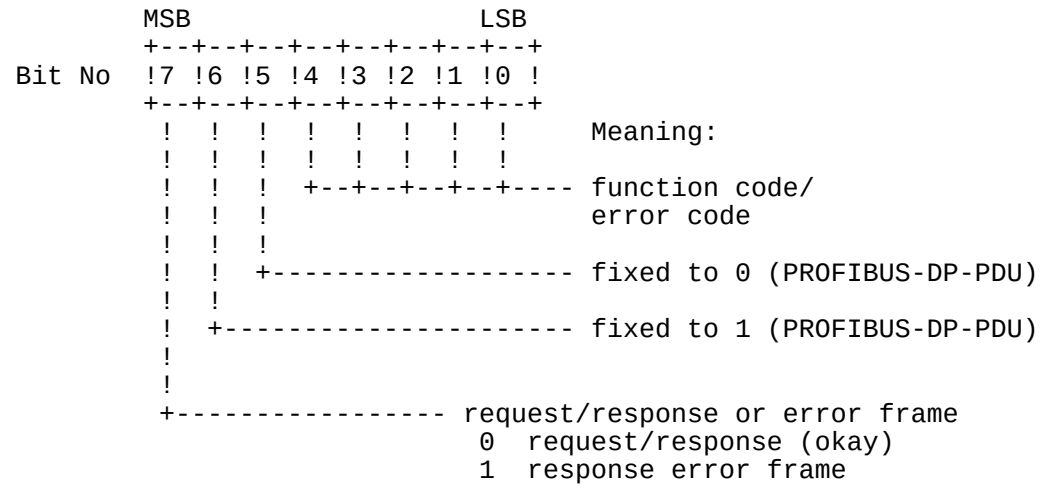


The maximum length of the Data_Unit is 246 octets according to the Data Link Layer Service Protocol Specifications. If additional information has to be transferred, the possible length of the user data is reduced.

11.1.3 Structure of Additional Information

The first octet of the additional information is the Function_Num.

The Function_Num has the following format:



Function_Num will be transferred in the request and response frame. If the function is correctly processed, bit 7 of Function_Num in the response frame is set to zero. In case of an error an error frame will be transferred. In this frame Function_Num contains an error code with bit 7 set to one.

In the following there is a list of the parameter Function_Num for the particular DP-Master functions.

function code (Bit 0 to 4 of Function_Num):

0	reserved
1	Get_Master_Diag
2	Start_Seq
3	Download
4	Upload
5	End_Seq
6	Act_Para_Brct
7	Act_Param
8 to 31	to be defined

error code:

0	reserved	
1	FE	
2	NE	
3	AD	
4	EA	Meaning of the error codes
5	LE	see table 7.
6	RE	
7	IP	
8	SC	
9	SE	
10	NE	
11	DI	
12 to 31	to be defined	

These error codes will be directly transferred by means of a response error frame. All other error codes are implicitly part of the FC of the response frame or are generated from the local FDL.
The transferred error codes in the response error frame will be delivered from the DDLM in the parameter Status to the User-Interface.

Table 43. Coding of the PDU (without address extension)

Function	Parameter
Get_Master_Diag.req	Function_Num,Identifier,
Get_Master_Diag.res	Function_Num,Identifier,Diagnosic_Data
Start_Seq.req	Function_Num,Area_Code,Timeout
Start_Seq.res	Function_Num,Area_Code,Timeout, Max_Len_Data_Unit
Download.req	Function_Num,Area_Code,Add_Offset, Data
Download.res	Function_Num,Area_Code,Add_Offset
Upload.req	Function_Num,Area_Code,Add_Offset, Data_Len
Upload.res	Function_Num,Area_Code,Add_Offset, Data
End_Seq.req	Function_Num
End_Seq.res	Function_Num
Act_Para_Brct.req	Function_Num,Area_Code
Act_Param.req	Function_Num,Area_Code,Activate
Act_Param.res	Function_Num,Area_Code,Activate

Example:

The coding of the PDU of the function **DDLM_Download** will clarify the coding.

Download(Area_Code 1, Add_Offset 3800 hex, Data (220 Octets))

Download.req PDU:
43H, 01H, 38H, 00H, Octet1, Octet2, .. Octet220

Download.res PDU: (without error)

43H, 01H, 38H, 00H

Download.res PDU: (with error code AD)

C3H

11.2 Boolean

Representation of the value true or false in an octet.

Notation: Boolean
Range of Value: true or false
Coding: false is represented by the value 00H
true is represented by the value FFH

Bit No	!7	!6	!5	!4	!3	!2	!1	!0
Octet 1	!1	!1	!1	!1	!1	!1	!1	!1

Figure 18. Representation of the value true

Bit No	!7	!6	!5	!4	!3	!2	!1	!0
Octet 1	!0	!0	!0	!0	!0	!0	!0	!0

Figure 19. Representation of the value false

11.3 Unsigned

Unsigned Values are unsigned quantities.

Notation: Unsigned8, Unsigned16, Unsigned32

Range of Value:

Datatype	Range of Value	Length
Unsigned8	0 to 255	1 Octet
Unsigned16	0 to 65535	2 Octet
Unsigned32	0 to 4294967295	4 Octet

Coding: Binary

Bit No	!7	!6	!5	!4	!3	!2	!1	!0
Octet 1	!2^{15}	!2^{14}	!2^{13}	!2^{12}	!2^{11}	!2^{10}	!2^{9}	!2^{8}
Octet 2	!2^{7}	!2^{6}	!2^{5}	!2^{4}	!2^{3}	!2^{2}	!2^{1}	!2^{0}

Figure 20. Representation of the Datatype Unsigned16

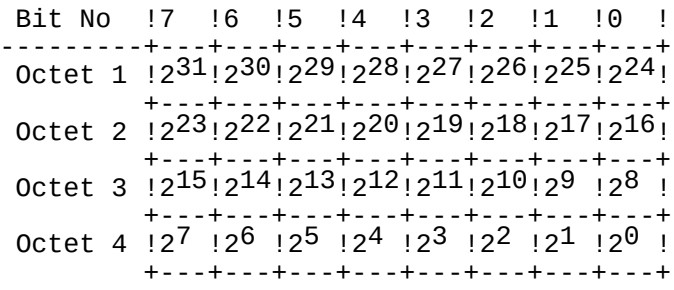


Figure 21. Representation of the Datatype Unsigned32

11.4 Octet-String

Notation: Octet-String
Coding: Binary

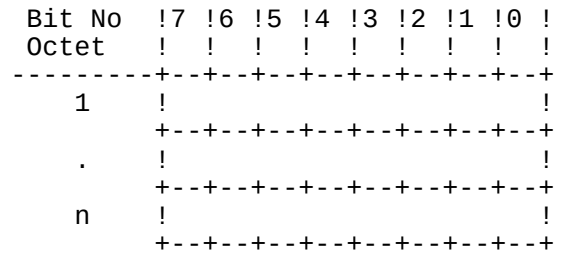


Figure 22. Representation of the Datatype Octet-String

11.5 Visible-String

Notation: Visible-String
Range of Value: see DIN 66003 and ISO 2375
Coding: see DIN 66003

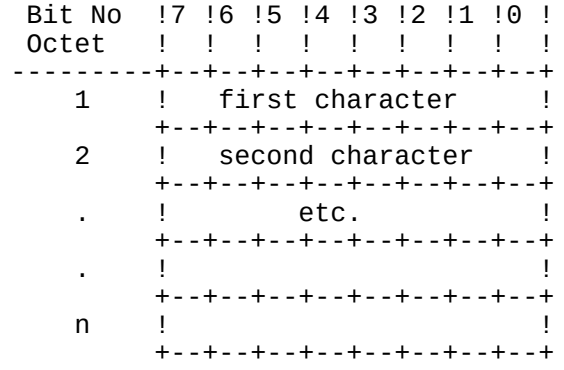


Figure 23. Coding of Data of the Data Type Visible-String

11.6 Coding of Set_Slave_Add and Global_Control

Table 44. Coding of the functions Set_Slave_Add and Global_Control

Function	Parameter
Set_Slave_Add.req	New_Slave_Add,Ident_Number,No_Add_Chg, Rem_Slave_Data
Global_Control.req	Control_Command,Group_Select

11.7 Structure of the Data_Unit at Data_Exchange

The structure of the Data_Unit for the function Data_Exchange is defined by the configuration identifiers which are passed to the DP-Slave at configuration.

The following Example shows how the data in the Data_Unit will be transferred, corresponding to the given configuration identifiers.

Configuration identifiers (C-ID):

1.C-ID	2.C-ID	3.C-ID	4.C-ID	5.C-ID	6.C-ID	7.C-ID	
1 B_I	1 W_I	2 W_O	2 B_IO	1 B_O	2 W_I	3 W_A	

1 B_I: 1 Byte input; 1 W_I: 1 Word input
1 B_O: 1 Byte output; 1 W_O: 1 Word output
2 B_IO: 2 Byte out- and input
etc.

The sequence of data in the Data_Unit is shown in figure 24.

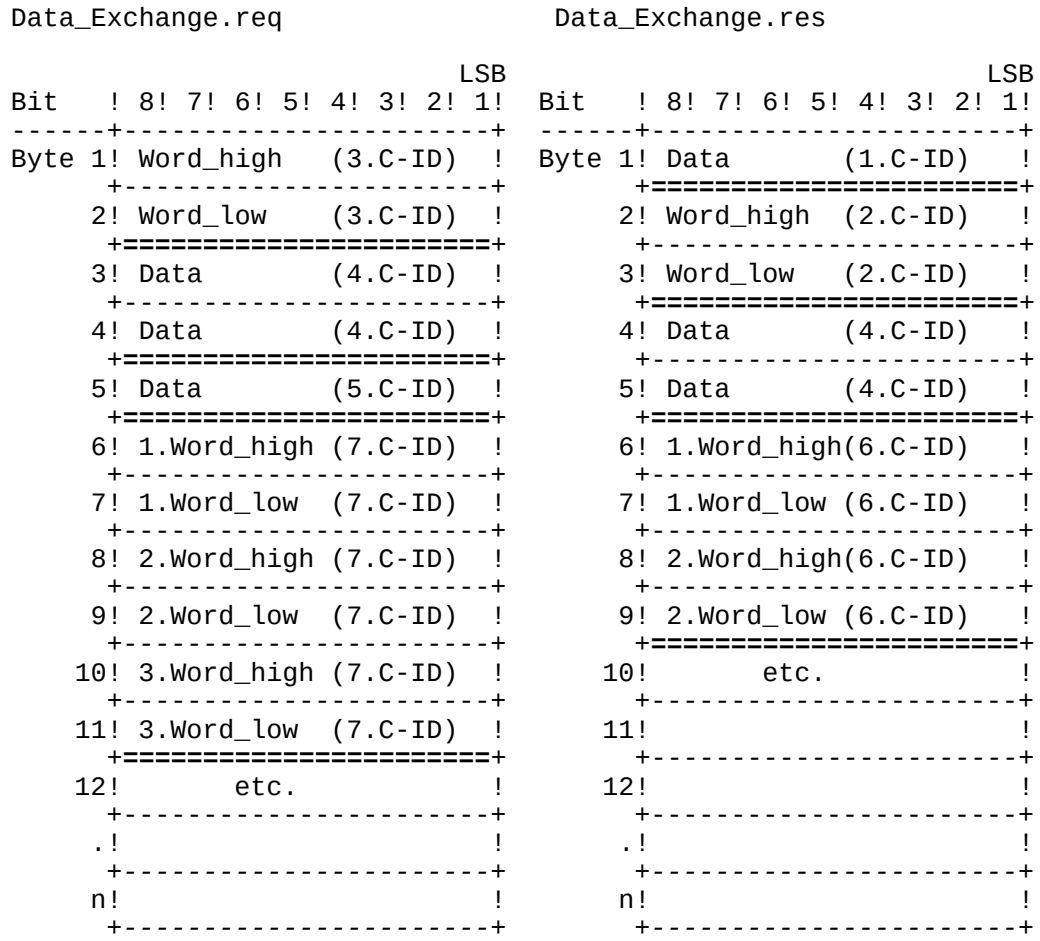


Figure 24. Structure of the Data_Unit for the Request- and Response-Frame

For the configuration identifier "free place" no data will be transferred in the Data_Unit.

11.8 Master-Parameter Set

The master parameter set contains the bus parameter set and DP-Slave parameter sets and is built as in section "Coding of the Bus Parameter Set" and "Coding of Parameter Set of DP-Slave" described.

11.8.1 Coding of the Bus Parameter Set

The structure of the bus parameter set is described in figure 25.

Name	Type
Bus_Para_Len	Unsigned16
FDL_Add	Unsigned8
Baud_rate	Unsigned8
T_{SL}	Unsigned16
min T_{SDR}	Unsigned16
max T_{SDR}	Unsigned16
T_{QUI}	Unsigned8
T_{SET}	Unsigned8
T_{TR}	Unsigned32
G	Unsigned8
HSA	Unsigned8
max_retry_limit	Unsigned8
Bp_Flag	Unsigned8
Min_Slave_Interval	Unsigned16
Poll_Timeout	Unsigned16
Data_Control_Time	Unsigned16
Octet 1 (reserved)	Octet-String
...	
Octet 6 (reserved)	
Master_User_Data_Len	Unsigned16
Master_Class2_Name	Visible-String(32)
Master_User_Data	Octet-String

Figure 25. Structure of the bus parameter set

Bus_Para_Len:

This parameter contains the length of Bus_Para inclusive the lengthparameter.
Range: 34 to 2^{16}-1

FDL_Add:

This parameter contains the own address of the DP-Master.
Range: 0 to 125

Baud_rate:

This parameter contains the code number for the baud rate.
Range:

```
  7 to 255 =>  reserved for further baud rates
         6 =>  1500  kbit/s
         4 =>   500  kbit/s
         3 =>   187,5kbit/s
         2 =>    93,75kbit/s
         1 =>    19,2kbit/s
         0 =>     9,6kbit/s
```

T_{SL}, min T_{SDR}, max T_{SDR}, T_{QUI}, T_{SET}, T_{TR}, G, HSA, max_retry_limit:

These parameters are described in the Data Link Layer Service Protocol Specifications.

Bp_Flag:

This parameter contains flags for the User-Interface

```
         MSB                  LSB
         +--+--+--+--+--+--+--+--+
 Bit No  !7 !6 !5 !4 !3 !2 !1 !0 !
         +--+--+--+--+--+--+--+--+
          !  !  !  !  !  !  !  !     Meaning:
          !  !  !  !  !  !  !  !
          !  +--+--+--+--+--+--+---- reserved
          !
          !
          +--- Error_Action_Flag
               0 no change of the operation mode in case of an error
               1 change of the operation mode in case of an error
```

Min_Slave_Interval:

Meaning: see section "Data Interface"
Range: 1 to 2^{16}-1
Timebase: 100µs

Poll_Timeout:

Meaning: see section "Data Interface"
Range: 1 to 2^{16}-1
Timebase: 1ms

Data_Control_Time:

Meaning: see section "Data Interface"
Range: 1 to 2^{16}-1
Timebase: 10ms

Master_User_Data_Len:

This parameter contains the length of Master_User_Data inclusive the lengthparameter.
Range: 2 to (2^{16}-33)

Master_Class2_Name:
This parameter indicates the name of the DP-Master (class 2) the master parameter set where created with. The length of this name shall not exceed 32 characters.

Master_User_Data:

This field contains specific data from the manufacturer which are necessary for the bus parameter set.

11.8.2 Coding of Parameter Set of DP-Slave

The structure of the DP-Slave parameter sets is described in figure 26. The FDL address of the corresponding DP-Slave is the Area_Code with which the DP-Slave parameter set is loaded.

Name	Type
Slave_Para_Len	Unsigned16
Sl_Flag	Unsigned8
Slave_Type	Unsigned8
Octet 1 (reserved)	Octet-String
...	
Octet 12 (reserved)	
Prm_Data_Len	Unsigned16
Prm_Data	Octet-String
Cfg_Data_Len	Unsigned16
Cfg_Data	Octet-String
Add_Tab_Len	Unsigned16
Add_Tab	Octet-String
Slave_User_Data_Len	Unsigned16
Slave_User_Data	Octet-String

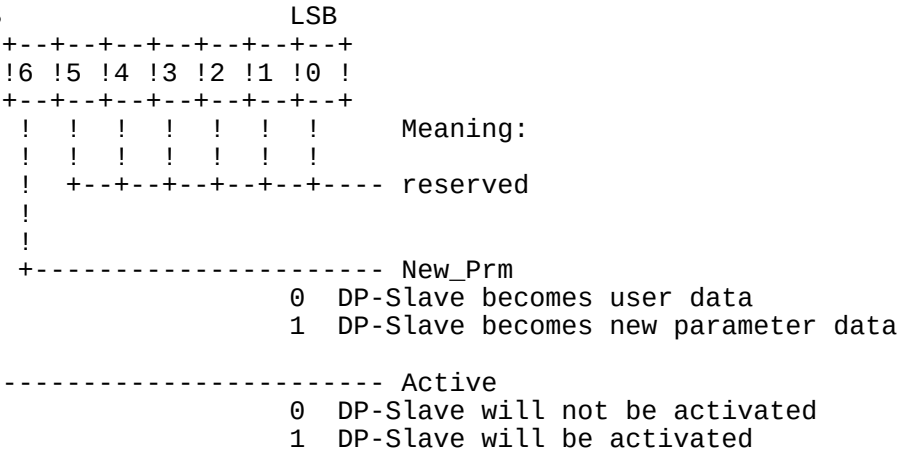
Figure 26. Structure of the DP-Slave parameter set

Slave_Para_Len:

This parameter contains the length of Slave_Para inclusive the lengthparameter. A DP-Slave parameter set can be deleted by setting the Slave_Para_Len to zero.
Range: 0 to 2^{16}-1

Sl_Flag:

This parameter contains slave specific flags.



Slave_Type:

This parameter contains a manufacturer specific type denotation for the DP-Slave.
Range: 0 to 2^{8}-1

0	DP-Slave
1 to 15	reserved
16 to 255	manufacturer specific

Prm_Data_Len:

This parameter contains the length of Prm_Data inclusive the lengthparameter.
Range: 9 to 246

Prm_Data:

This parameter is described in section "Send Parameter Data".

Cfg_Data_Len:

This parameter contains the length of Cfg_Data inclusive the lengthparameter.
Range: 3 to 246

Cfg_Data:

This parameter is described in section "Check Configuration Data".

Add_Tab_Len:

This parameter contains the length of Add_Tab inclusive the lengthparameter.
Range: 2 to 2^{16}-31

Add_Tab:

This parameter contains the address-assignment-table of the DP-Slave. In the case of a PLC the list Add_Tab contains the PLC addresses of the decentralized addresses.

Slave_User_Data_Len:

This parameter contains the length of Slave_User_Data inclusive the lengthparameter.
Range: 2 to 2^{16}-31

Slave_User_Data:

This field contains manufacturer specific data that characterize the DP-Slave for the DP-Master.

11.9 Coding of Statistic Counters

The statistic counters are described in figure 27.

Frame_sent_count_0	Unsigned32
Error_count_0	Unsigned16
Frame_sent_count_1	Unsigned32
Error_count_1	Unsigned16
. . .	
Frame_sent_count_126	Unsigned32
Error_count_126	Unsigned16
SD_count	Unsigned32
SD_error_count	Unsigned16

Figure 27. Coding of statistic counters

The index of the counters corresponds to the station address of the participants.
The statistic counters can be rest by means of the function DDLM_Download. For this the Area_Code 129 (and the corresponding Add_Offset) with data equal to zero has to be addressed. To read the statistic counters the function DDLM_Upload has to be used.

For Add_Offset and Data_Len the following rules are valid:
Add_Offset = n·6; n = 0 to 127
Data_Len = n·6; n = 1 to 39

12 Direct-Data-Link-Mapper (DDLM)

12.1 General

As already mentioned in the description of the protocol architecture, the Direct-Data-Link-Mapper (DDLM) makes the connection between the User-Interface and layer 2. All functions which are passed at the interface of the User-Interface are mapped by the DDLM to the FDL and FMA1/2 services of layer 2. The DDLM delivers the necessary layer 2 parameters of the function call (SSAP, DSAP, Serv_class, ...) for layer 2, receives the confirmations and indications from layer 2 and delivers them to the User-Interface.

12.2 Interface between DDLM and Layer 2

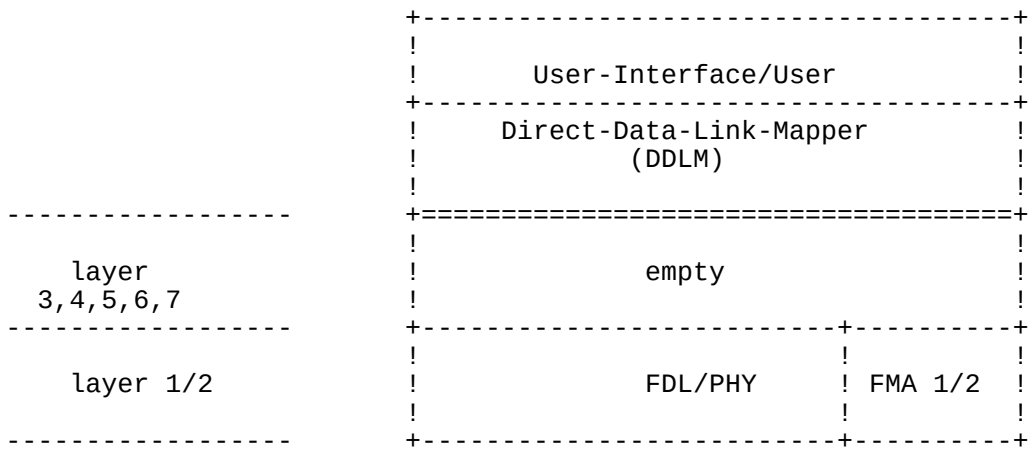


Figure 28. Interface between DDLM and layer 2

The DDLM is directly put upon the FDL user - FDL interface and on the FMA1/2 user - FMA1/2 interface as described in the Data Link Layer Service Definitions.

The DDLM uses for the mapping of the DDLM functions on the layer 2 the following services.

a) FDL services:

- Send and Request Data with Reply (SRD)
- Send Data with No Acknowledge (SDN)

b) FMA1/2 services:

- Reset FMA1/2
- Set Value FMA1/2
- Read Value FMA1/2
- Event FMA1/2
- SAP Activate FMA1/2
- RSAP Activate FMA1/2
- SAP Deactivate FMA1/2

The mapping of the DDLM functions to layer 2 and the associated Service Access Points (SAP) are shown in table 45:

Table 45. Assignment of FDL parameters to the primitives at master-slave communication

DDLM_Primitive	SSAP	DSAP	Layer 2 Service	Serv_class
DDLM_Data_Exchange.req/.ind	NIL	NIL	SRD	high
DDLM_Data_Exchange.con				low/high
DDLM_Data_Exchange_Upd.req				low/high
DDLM_Chk_Cfg.req/.ind	62	62	SRD	high
DDLM_Chk_Cfg.con				
DDLM_Set_Prm.req/.ind	62	61	SRD	high
DDLM_Set_Prm.con				
DDLM_Slave_Diag.req/.ind	62	60	SRD	high
DDLM_Slave_Diag.con				low
DDLM_Slave_Diag_Upd.req				low
DDLM_Get_Cfg.req	62	59	SRD	high
DDLM_Get_Cfg.con				low
DDLM_Get_Cfg_Upd.req				low
DDLM_Global_Control.req/.ind	62	58	SDN	high
DDLM_RD_Outp.req	62	57	SRD	high
DDLM_RD_Outp.con				low
DDLM_RD_Outp_Upd.req				low
DDLM_RD_Inp.req	62	56	SRD	high
DDLM_RD_Inp.con				low
DDLM_RD_Inp_Upd.req				low
DDLM_Set_Slave_Add.req/.ind	62	55	SRD	high
DDLM_Set_Slave_Add.con				

Table 46. Assignment of FDL parameters to the primitives at master-master communication

DDLM_Primitive	SSAP	DSAP	Layer 2 Service	Serv_ class
DDLM_xxx.req/.ind	54	54	SRD	low
DDLM_xxx.con				low
DDLM_xxx_Upd.req				low
DDLM_Act_Para_Brct.req/.ind	54	54	SDN	low
DDLM_Act_Para_Brct.con				

The functions for master-master communication in the requester and also in the responder operate all at the Service Access Point 54 (Master-Master-SAP).

12.3 Communication between DP-Master and DP-Slave

The communication between DP-Master and DP-Slave is described in figures 7 to 10 (see section "Master-Slave Communication").
If the DP-Master wants to communicate with a DP-Slave, the DP-Master requests the Diag_Data of the DP-Slave, in order to check the readiness for operation of the DP-Slave.
This function request (DDLM_Slave_Diag) will be repeated until the DP-Slave responds with the requested data.
If the DP-Slave responds with the requested diagnostic information, the DP-Master checks whether another DP-Master occupies this DP-Slave. If this is not the case, the DP-Slave will be initialized and configured (DDLM_Set_Prm/DDLM_Chk_Cfg).
Subsequently the DP-Master requests again the diagnostic data from the DP-Slave. The following messages can occur:

a) parametrization and configuration error
b) The DP-Slave is occupied by another DP-Master
c) Static user diagnostics, so normal data transfer makes no sense
   (for Example: external power supply for the outputs is missing)
d) DP-Slave is not ready for operation

In the cases a) and b) the DP-Master returns to the starting state and checks again the readiness for operation of the DP-Slave.
In the cases c) and d) the DP-Master requests the diagnostic information from the DP-Slave until the mentioned messages disappear.

If no error messages occur the DP-Master starts the user data exchange to the DP-Slave. The input and output data will be transferred to and from the DP-Slave. The DP-Master (class 1) indicates his own operation mode to the DP-Slaves for the first time.

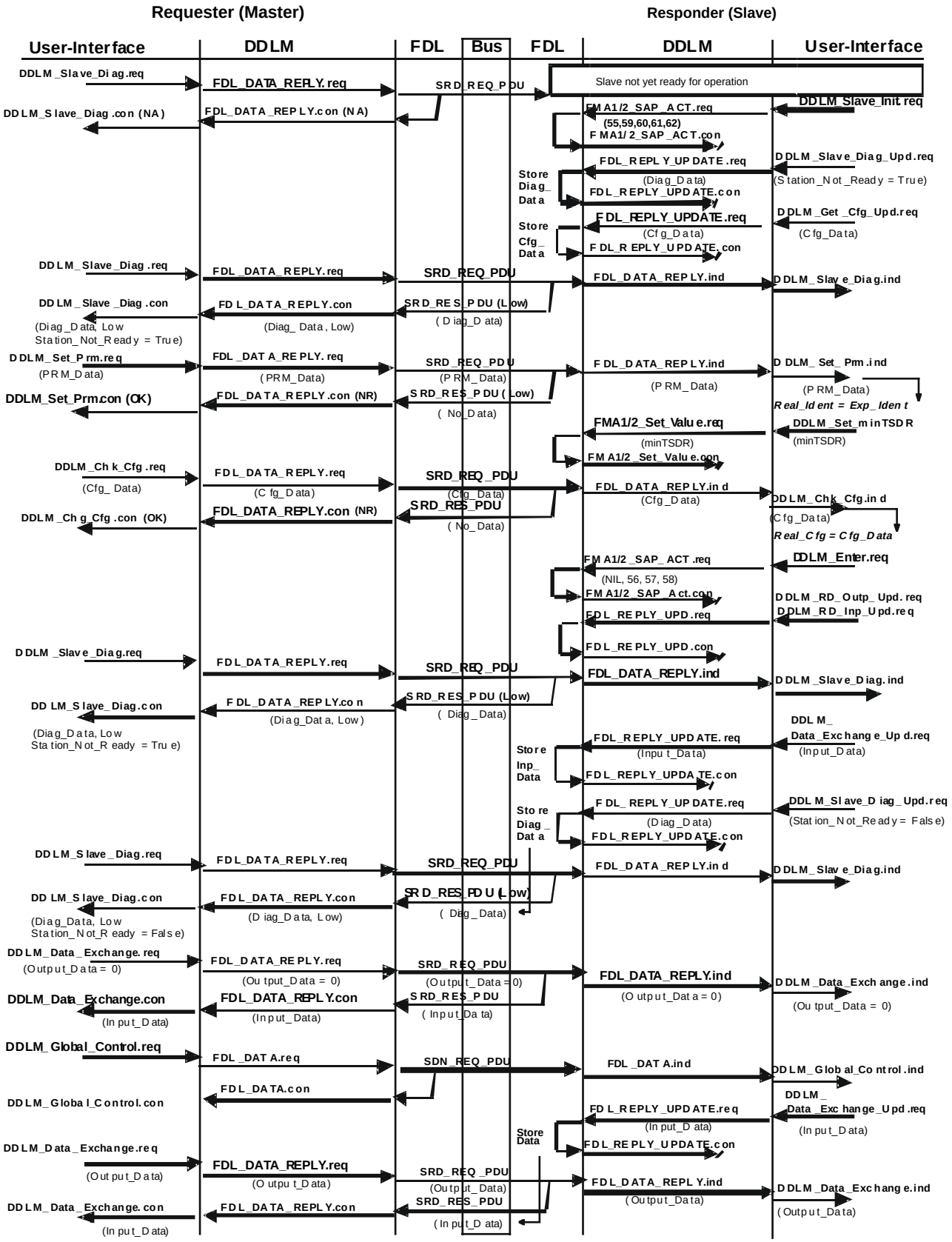


Figure 29. Sequence of the communication between DP-Master and DP-Slave

12.4 Communication between DP-Master and DP-Master

The communication between DP-Master (class 2) and DP-Master (class 1) is handled in a fixed way (see figure 30) with one exception.
This exception is the function **DDLM_Act_Para_Brct**, which is transferred as a multicast from the DP-Master (class 2) to the corresponding DP-Master (class 1).

Communication is always initiated by the DP-Master (class 2). The DP-Master (class 2) sends a request to the DP-Master (class 1) and waits for the response. The service is processed by the DDLM of the DP-Master (class 2). After the receipt of the response the DP-Master (class 2) can make a new request.
At one point of time the DP-Master (class 1) can only communicate with one DP-Master (class 2).
The DP-Master communication is initiated by a DDLM_xxx.req (see also section "DP-Master - DP-Master Functions") from the DP-Master (class 2). The DDLM passes this request to layer 2 and checks the SRD_RES_PDU after receipt of the response. If the response status is NR (no reply data) it will retry once more with a new SRD with no L_sdu to get response data. This procedure will be repeated by the DP-Master (class 2) until
- an erroneous SRD_RES_PDU
- or a SRD_RES_PDU with response data is received,
- or the watchdog T1 expires.

The result will then be passed to the User of the DP-Master (class 2) by means of a corresponding DDLM_xxx.con.

The timer T1 will be started after receipt of the DDLM_xxx.req in the DDLM of the DP-Master (class 2) and will be stopped after release of the DDLM_xxx.con. The value loaded in the timer T1 depends on the FDL parameter and the performance of the DP-Master (class 1).

The responder (DP-Master (class 1)) waits after a reset for a request. If the DDLM of the DP-Master (class 1) receives a valid request, this request is passed to the user by the corresponding DDLM_xxx.ind. At this point of time an access protection was set to the local service access point (SAP) for the communication with the DP-Master (class 2). This is necessary to make sure that the response data will be sent to the right DP-Master (class 2).

After receipt of the response from the user (DDLM_xxx.res) the response data will be passed with a FDL_REPLY_UPDATE.req to layer 2. Additionally, a timer will be started which controls the request for the response. If this timer expires and the response data are still not fetched by the DP-Master (class 2), the DDLM deletes the response data because the DDLM presumes that the DP-Master (class 2) is no longer available. In this case the access protection for the DP-Master (class 2) will be cancelled. This will also be done in the case of a single transaction if a FDL_DATA_REPLY.ind indicates that the response data are fetched by the DP-Master (class 2). The value loaded in the timer of the DP-Master (class 1) depends on the FDL parameter and the performance of the DP-Master (class 2).

In case of master-master communication a sequence of requests can put into parentheses. Therefore the access protection will be set by the "DDLM_Start_Seq" and released by the "DDLM_End_Seq". To avoid errors the time between two requests will be supervised. The value for the time supervision is added to the DDLM_Start_Seq request. With the DDLM_Start_Seq.req the user can additionally send an areacode. By means of this areacode at the beginning of a sequence an area can be reserved at the user in the DP-Master (class 1).

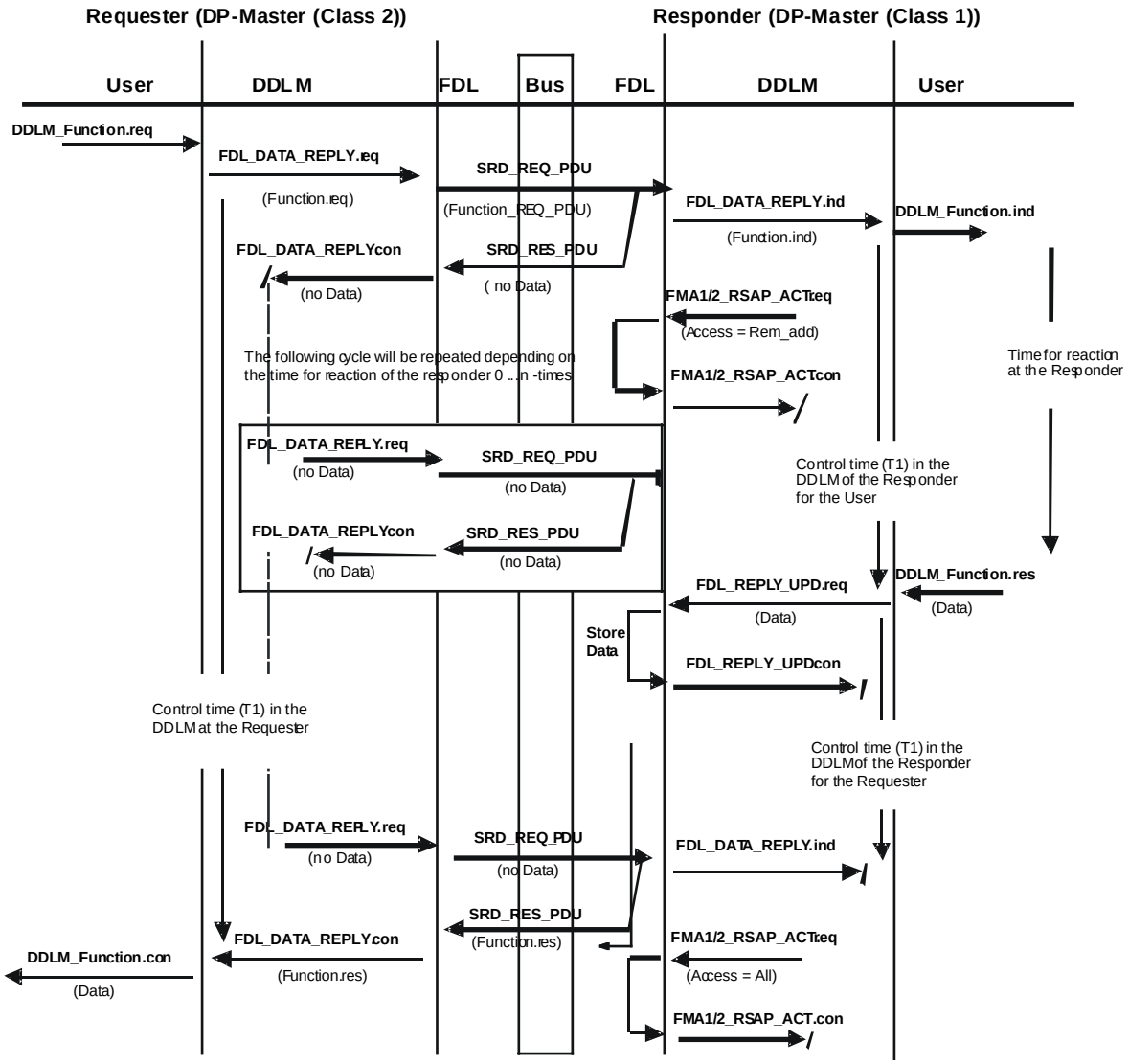


Figure 30. Sequence of communication between DP-Master (class˚2) and DP-Master (class˚1)

13 Formal Description of State Machines

13.1 General

The formal description is based upon a model. This model describes the DP protocol with the help of state machines (represented by state diagrams). The protocol sequences are described by different states (represented by ellipses with the name of the state) and transitions between states (represented by lines with arrows). State changes are caused by events (for Example time out) combined with conditions. The state changes in turn lead to actions or reactions.
The formal description follows the description of the LLI state machines in the PROFIBUS Specification. This description is based on an abstract model and it is not the intention of the model to prescribe the translation in a concrete realization.

13.2 Communication Model of DP-Master (Class 1)

13.2.1 General

The DP-Master (class 1) is subdivided in 4 instances:

1. layer 1/2, Subset of PROFIBUS (FDL and FMA1/2)
2. Direct-Data-Link-Mapper (DDLM)
3. User-Interface
4. User

An overview of the interactions between the individual instances is given in figure 31.

13.2.2 FDL and FMA1/2

The layer 2 has to handle at least the FDL services SRD and SDN. In addition to the services Reset FMA1/2 and Event FMA1/2 the service SAP Activate FMA1/2 is also mandatory. To enable the communication with a DP-Master (class 2), it is necessary to additionally implement the SRD-/SDN-responder functions and the service RSAP Activate FMA1/2. In this case layer 2 must be able to handle SAP access protection and Reply-Update-Mode = single.
Additional hints are given in the sections "Transmission Technology" and "Medium Access and Transmission Protocol".

13.2.3 Direct-Data-Link-Mapper

The DDLM contains a mapping of the DDLM functions onto the layer 2 services.

For the DDLM of the DP-Master (class 1) the following local functions are mandatory:

-DDLM_Master_Init
-DDLM_Responder_Init
-DDLM_Fault
-DDLM_Event

Additionally the following data transmission functions are also mandatory:

-DDLM_Slave_Diag
-DDLM_Set_Prm
-DDLM_Chk_Cfg
-DDLM_Data_Exchange
-DDLM_Global_Control

With these functions the User-Interface carries out the data exchange with the DP-Slaves.

In case of communication between a DP-Master (class 2) and a DP-Master (class 1) the DDLM works transaction oriented. The DP-Master (class 1) can at one point of time communicate with at most one DP-Master (class 2). It is not allowed to have parallel services active.
The DP-Master (class 2) initiates these functions for the master-master communication. It is not necessary for the DP-Master (class 1) to confirm these functions with an immediate response. In this case the DP-Master (class 2) polls the DP-Master (class 1) until the DP-Master (class 1) provides the response data (see figure 30).

For the DDLM of the DP-Master (class 1) the following function is mandatory:

-DDLM_Get_Master_Diag

The DDLM provides optionally the following functions for the master- master communication:

-DDLM_Start_Seq
-DDLM_End_Seq
-DDLM_Upload
-DDLM_Download
-DDLM_Act_Para_Brct
-DDLM_Act_Param

13.2.4 User-Interface

The User-Interface communicates with the DDLM using the functions as described in the sections before.

The interface to the user is formed as a data and service interface. The interface between user and User-Interface of the DP-Master (class 1) is described in detail in section "DP-Master (class 1)".

The User-Interface consists of three function blocks:

- Slave-Handler
- Scheduler
- Service-Handler

The state machine of the Slave-Handler takes over the control of exactly one DP-Slave. The management of each DP-Slave is made independently. The state machines for each DP-Slave react individually depending on the state of the DP-Slave.
The DP-Slaves are processed following a fixed sequence. At each call the corresponding Slave-Handler will make only one DDLM request. The Scheduler determines the sequence of the calls of the corresponding Slave-Handlers (and therefore the sequence of telegrams on the network). A given period of time between two slave poll cycles has to be secured by the Scheduler. This period of time (Min_Slave_Interval) is stored in the master parameter set. This time depends on the performance of the used DP-Slaves in a given system.

The Scheduler additionally has to initialize the DDLM and to reset the Slave-Handlers at power up. Locally generated events and errors from FDL and DDLM are evaluated by the Scheduler, too.
The Scheduler also controls the global state transitions:

- Offline:
  The User-Interface carries out no actions, DDLM functions are suspended.

- Stop:
  The Slave-Handlers are locked, DDLM functions for master- master communication (e. g. to load the parameters) are handled.

- Clear:
  The Slave-Handlers are active, output data equal to zero are transferred to the DP-Slaves.

- Operate:
  The Slave-Handlers are active, the output data from the user are transferred to the DP-Slaves.

The User-Interface takes over protective functions (Auto-Clear), if a DP-Slave breaks down during operation. The Scheduler clears the outputs of the DP-Slaves if the Error_Action_Flag is set and changes the operation mode to the state "Clear". This change of the operation mode will not take place if the Error_Action_Flag in the bus parameter set is not set.
The Mark function supports the user to handle the Sync and Freeze control commands. This function gives the user the possibility to wait for a complete slave poll cycle after which a defined transfer of input and output data can take place.

The Service-Handler takes over the control for the local functions Load_Bus_Par, Read_Value and Delete_SC.

13.2.5 User

The user represents the real application process of the corresponding device and has to be defined depending on the application.

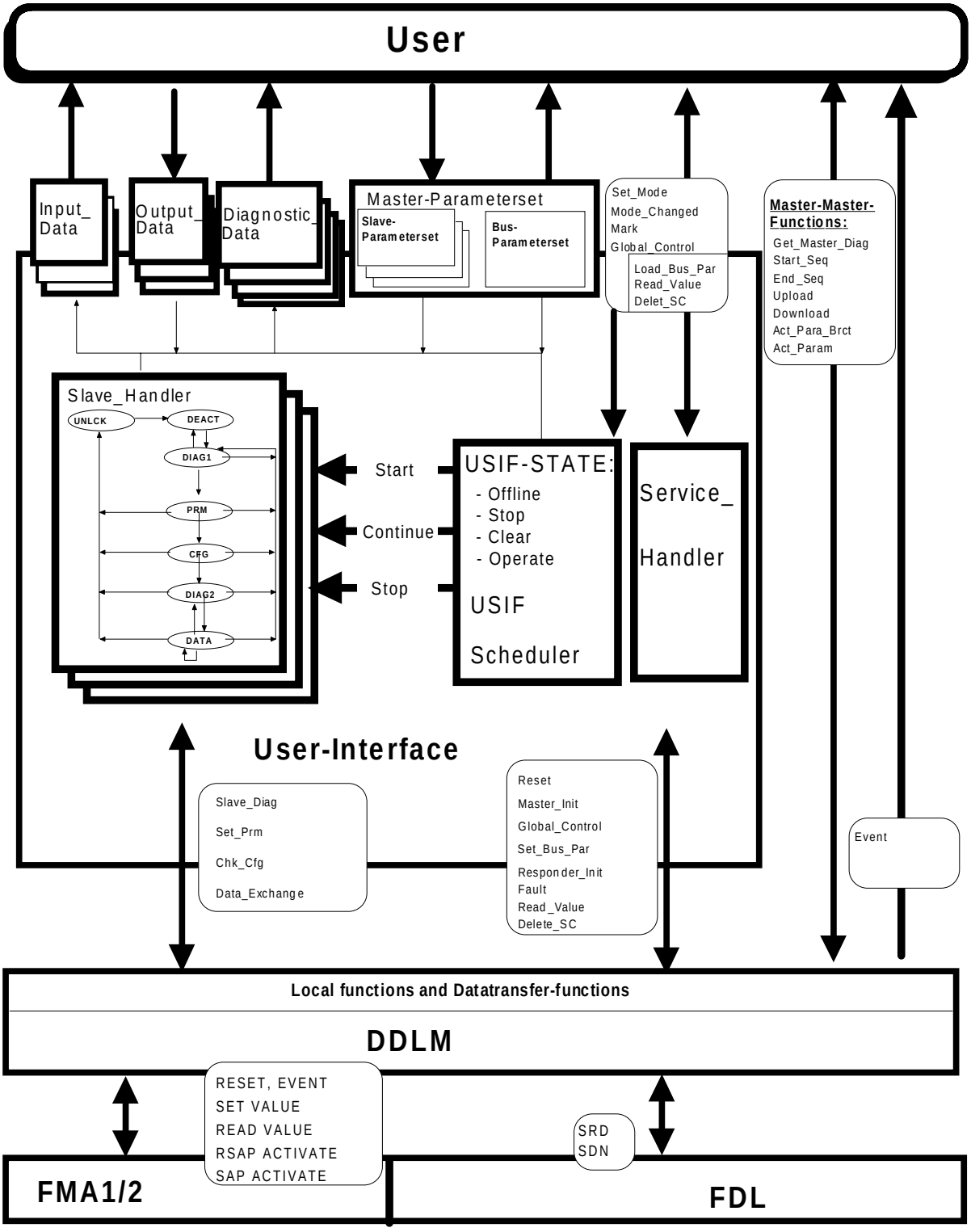


Figure 31. Structure of a DP-Master (class 1)

13.3 State Machines in the DP-Master (Master-Slave)

13.3.1 State Machine Slave-Handler

13.3.1.1 State Diagram Slave-Handler

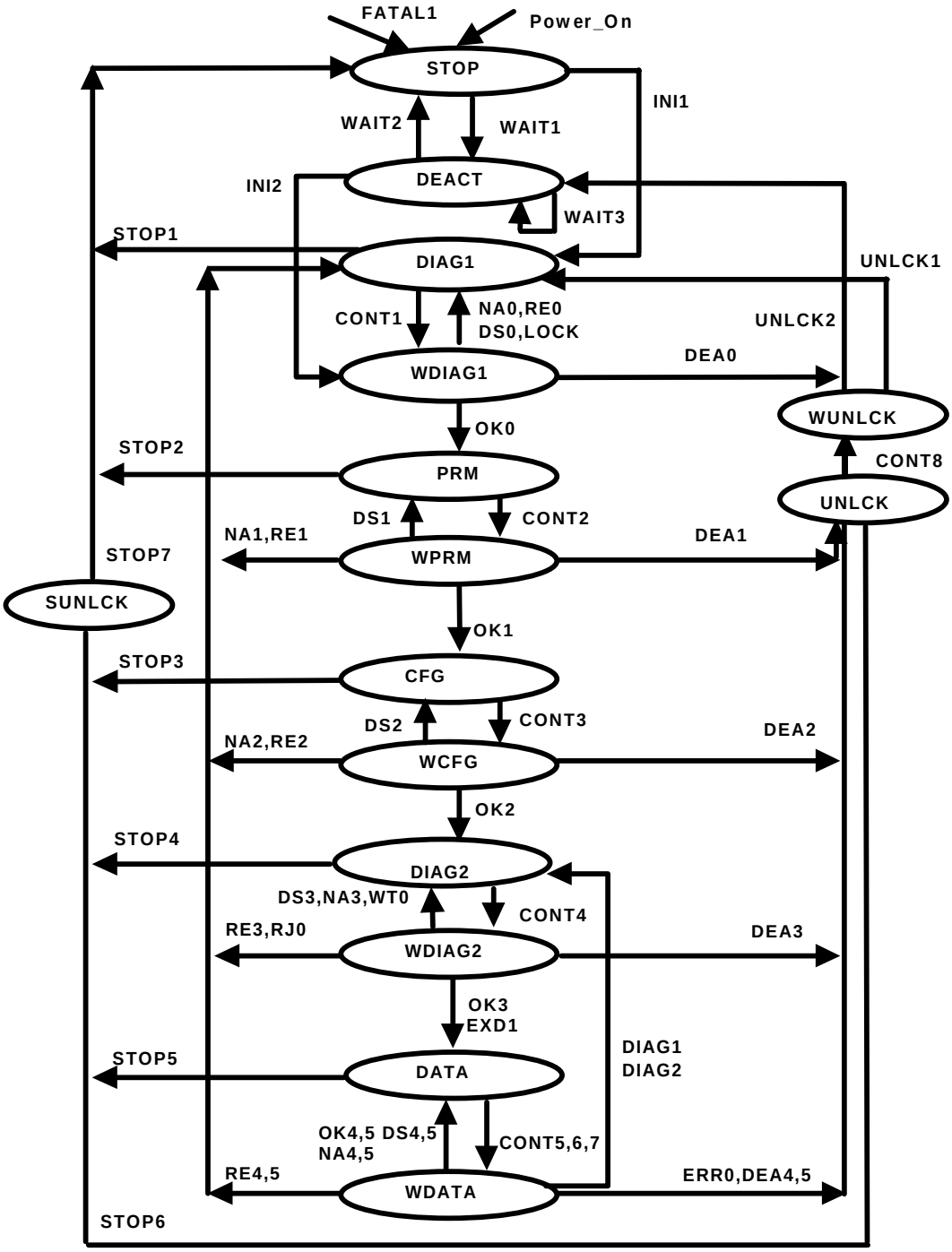


Figure 32. State Diagram of the Slave-Handler

13.3.1.2 State Machine Description

For each possible DP-Slave a state machine (Slave-Handler) is established and started by a superior state machine, the Scheduler. The Scheduler controls the Slave-Handler through the primitives:

- **Start_Slave_Handler.req**
- **Continue_Slave_Handler.req (Output_Clear)**
  The parameter Output_Clear (Boolean) indicates that not the output data but zero has to be sent.

- **Stop_Slave_Handler.req**

The Slave-Handler indicates the result of the processing by means of the following primitives to the Scheduler:

- **Start_Slave_Handler.con**
- **Continue_Slave_Handler.con (Diag)**
  The parameter Diag (Boolean) indicates whether a regular data exchange has taken place or not.

This parameter has the value false if DDLM_Data_Exchange was performed successfully for a DP-Slave that is marked as active in the DP-Slave parameter set. This bit is true in all other cases.
This parameter is always true for DP-Slaves which are marked in the DP-Slave parameter set as not active.

- **Stop_Slave_Handler.con**

The Slave-Handlers additionally communicate with the user via the data interface.

The Slave-Handler manages the individual states for every DP-Slave. The following main states will be distinguished:

- diagnostic
- parametrization
- configuration
- data exchange

The user and the Slave-Handler communicate via the data interface using the parameter a) to g):

Legend:
-r will only be read by the Slave-Handler and written by the User
-w will only be written by the Slave-Handler and read by the User

a) Active
(Sl_Flag.7) -r
Indicates whether the DP-Slave should be activated (=True) or not. In the second case the state machine do not reach the state DATA.

b) New_Prm
(Sl_Flag.6) -r/-w
Indicates whether new parameter data are available for the DP-Slave.

c) Prm_Data
(Octet-String) -r
Parameter data of the DP-Slave. Structure see section "Send Parameter Data" of the interface DDLM - User-Interface.

d) Cfg_Data
(Octet-String) -r
Configuration data of the DP-Slave. Structure see section "Check Configuration Data" of the interface DDLM - User-Interface.

e) Output_Data
(Octet-String) -r
Output data for the DP-Slave.

f) Input_Data
(Octet-String) -w
Input data for the DP-Slave.

g) Diag_Data
(Octet-String) -w
Diagnostic information of the DP-Slave. Structure see section "Read DP-Slave Diagnostic Information".

The Slave-Handler changes the following flags of the diagnostic information:

Diag.Invalid_Slave_Response
Diag.Station_Non_Existent
Diag.Deactivated
System_Diagnostic[Slave_Address]

The remaining diagnostic information will be transferred directly in the Diag_Data at the data interface after each DDLM_Slave_Diag.con.

The Slave-Handler uses the following internal variables. These variables are preset at initialization of the device and are used by the Slave-Handler.

a) exp_Inp_len
(Unsigned8)
Specifies the number of input data octets which are expected from the DP-Slave.

b) Internal_Data_Transfer_List[x]
(Boolean)
The Slave-Handler marks in this list whether a regular Data_Exchange function could be executed with the corresponding DP-Slave.

13.3.1.3 State Transitions

Power-On	**PON**	**STOP**
STOP	**WAIT1**	**DEACT**

Start_Slave_Handler.req
\Sl_Para_Exist=False OR Active=False
=> Start_Slave_Handler.con

STOP **INI1** **DIAG1**
Start_Slave_Handler.req
\Sl_Para_Exist=True AND Active=True
=> System_Diagnostic[x]=1
Diag.Invalid_Slave_Response = False
Diag.Deactivated = False
Start_Slave_Handler.con

DEACT **WAIT2** **STOP**
Stop_Slave_Handler.req
=> Stop_Slave_Handler.con

DEACT **WAIT3** **DEACT**
Continue_Slave_Handler.req(Output_Clear)
\Sl_Para_Exist=False OR Active=False
=> Continue_Slave_Handler.con(Diag=False)

DEACT **INI2** **WDIAG1**
Continue_Slave_Handler.req(Output_Clear)
\Sl_Para_Exist=True AND Active =True
=> System_Diagnostic[x] = 1
Diag.Invalid_Slave_Response = False
Diag.Deactivated = False
DDLM_Slave_Diag.req
Continue_Slave_Handler.con(Diag=True)

DIAG1 **CONT1** **WDIAG1**
Continue_Slave_Handler.req(Output_Clear)
=> DDLM_Slave_Diag.req

DIAG1 **STOP1** **STOP**
Stop_Slave_Handler.req
=> Diag.Deactivated = True
Stop_Slave_Handler.con

WDIAG1 **DEA0** **DEACT**
DDLM_Slave_Diag.con
\Active=False
=> Diag.Deactivated = True
System_Diagnostic[x] = 0
Continue_Slave_Handler.con(Diag=False)

WDIAG1 **NA0** **DIAG1**
DDLM_Slave_Diag.con
\Active=True AND Status=NA
=> Diag.Station_Non_Existent = True
Continue_Slave_Handler.con(Diag=True)

WDIAG1 **RE0** **DIAG1**
DDLM_Slave_Diag.con
\Active=True AND Status=RE/RS/NR/UE
=> Diag.Invalid_Slave_Response = True
Diag.Station_Non_Existent = False
Continue_Slave_Handler.con(Diag=True)

WDIAG1 **DS0** **DIAG1**
DDLM_Slave_Diag.con
\Active =True AND Status=DS
=> Continue_Slave_Handler.con(Diag=True)

WDIAG1 **LOCK** **DIAG1**
DDLM_Slave_Diag.con
\Active=True AND Status=OK AND Master_Lock=True
=> Continue_Slave_Handler.con(Diag=True)

WDIAG1 **OK0** **PRM**
DDLM_Slave_Diag.con
\Active=True AND Status=OK AND Master_Lock=False
=> New_Prm = False
Continue_Slave_Handler.con(Diag=True)

PRM **CONT2** **WPRM**
Continue_Slave_Handler.req(Output_Clear)
=> DDLM_Set_Prm.req(Prm_Data: Lock_Req=True,
Unlock_Req=False)

PRM **STOP2** **STOP**
Stop_Slave_Handler.req
=> Diag_Deactivated = True
Stop_Slave_Handler.con

WPRM **DEA1** **UNLCK**
DDLM_Set_Prm.con
\Active=False
=> Continue_Slave_Handler.con(Diag=False)

WPRM **DS1** **PRM**
DDLM_Set_Prm.con
\Active=True AND Status=DS
=> Continue_Slave_Handler.con(Diag=True)

WPRM **NA1** **DIAG1**
DDLM_Set_Prm.con
\Active=True AND Status=NA
=> Diag.Station_Non_Existent = True
Continue_Slave_Handler.con(Diag=True)

WPRM **RE1** **DIAG1**
DDLM_Set_Prm.con
\Active=True AND Status=RE/RS/RR/UE
=> Diag.Invalid_Slave_Response = True
Continue_Slave_Handler.con(Diag=True)

WPRM **OK1** **CFG**
DDLM_Set_Prm.con
\Active=True AND Status=OK
=> Continue_Slave_Handler.con(Diag=True)

CFG **CONT3** **WCFG**
Continue_Slave_Handler.req(Output_Clear)
=> DDLM_Chk_Cfg.req(Cfg_Data)

CFG **STOP3** **SUNLCK**
Stop_Slave_Handler.req
=> DDLM_Set_Prm.req(Prm_Data: Unlock_Req=True)

WCFG **DEA2** **UNLCK**
DDLM_Chk_Cfg.con
\Active=False
=> Continue_Slave_Handler.con(Diag=False)

WCFG **DS2** **CFG**
DDLM_Chk_Cfg.con
\Active=True AND Status=DS
=> Continue_Slave_Handler.con(Diag=True)

WCFG **NA2** **DIAG1**
DDLM_Chk_Cfg.con
\Active=True AND Status=NA
=> Diag.Station_Non_Existent = True
Continue_Slave_Handler.con(Diag=True)

WCFG **RE2** **DIAG1**
DDLM_Chk_Cfg.con
\Active=True AND Status=RE/RS/RR/UE
=> Diag.Invalid_Slave_Response = True
Continue_Slave_Handler.con(Diag=True)

WCFG **OK2** **DIAG2**
DDLM_Chk_Cfg.con
\Active=True AND Status=OK
=> Continue_Slave_Handler.con(Diag=True)

DIAG2 **CONT4** **WDIAG2**
Continue_Slave_Handler.req(Output_Clear)
=> DDLM_Slave_Diag.req

DIAG2 **STOP4** **SUNLCK**
Stop_Slave_Handler.req
=> DDLM_Set_Prm.req(Prm_Data: Unlock_Req=True)

WDIAG2 **DEA3** **UNLCK**
DDLM_Slave_Diag.con
\Active=False
=> Input_Data=0
Continue_Slave_Handler.con(Diag=False)

WDIAG2 **DS3** **DIAG2**
DDLM_Slave_Diag.con
\Active=True AND Status=DS
=> Continue_Slave_Handler.con(Diag=True)

WDIAG2 **NA3** **DIAG2**
DDLM_Slave_Diag.con
\Active=True AND Status=NA
=> Diag.Station_Non_Existent = True
System_Diagnostic[x] = 1
Continue_Slave_Handler.con(Diag=True)

WDIAG2 **RE3** **DIAG1**
DDLM_Slave_Diag.con
\Active=True AND Status=RE/RS/RR/UE
=> Input_Data=0
Diag.Station_Non_Existent = False
Diag.Invalid_Slave_Response = True
System_Diagnostic[x] = 1
Continue_Slave_Handler.con(Diag=True)

WDIAG2 **RJ0** **DIAG1**

DDLM_Slave_Diag.con

```
\Active=True AND Status=OK AND
 (Prm_Fault=True OR Cfg_Fault = True Or Prm_Req = True
  OR Master_Lock=True)
=>  Input_Data=0
    System_Diagnostic[x] = 1
    Continue_Slave_Handler.con(Diag=True)
```

WDIAG2 **WT0** **DIAG2**

DDLM_Slave_Diag.con

```
\Active=True AND Status=OK AND Prm_Fault=False
 AND Cfg_Fault=False AND Prm_Req=False                AND Master_Lock=False
 AND (Station_Not_Ready = True OR Stat_Diag=True)
=>  System_Diagnostic[x] = 1
    Continue_Slave_Handler.con(Diag=True)
```

WDIAG2 **OK3** **DATA**

DDLM_Slave_Diag.con

```
\Active=True AND Status=OK AND Prm_Fault=False
 AND Cfg_Fault=False AND Prm_Req=False
 AND Station_Not_Ready=False AND Master_Lock=False
 AND Stat_Diag=False       AND Ext_Diag=False
=>  System_Diagnostic[x]=0
    Continue_Slave_Handler.con(Diag=True)
```

WDIAG2 **EXD1** **DATA**

DDLM_Slave_Diag.con

```
\Active=True AND Status=OK AND Prm_Fault=False
 AND Cfg_Fault=False AND Prm_Req=False
 AND Station_Not_Ready=False AND Master_Lock=False
 AND Stat_Diag=False       AND Ext_Diag=True
=>  System_Diagnostic[x]=1
    Continue_Slave_Handler.con(Diag=True)
```

DATA **CONT5** **WDATA**

Continue_Slave_Handler.req(Output_Clear)

```
\Output_Clear=True AND New_Prm=False
=>  DDLM_Data_Exchange.req(Outp_Data=0)
```

DATA **CONT6** **WDATA**

Continue_Slave_Handler.req(Output_Clear)

```
\Output_Clear=False AND New_Prm=False
=>  DDLM_Data_Exchange.req(Outp_Data=Output_Data)
```

DATA **CONT7** **WDATA**

Continue_Slave_Handler.req(Output_Clear)

```
\New_Prm=True
=>  DDLM_Set_Prm.req(Prm_Data: Lock_Req=True,
                               Unlock_Req=False)
```

DATA **STOP5** **SUNLCK**

Stop_Slave_Handler.req

```
=>  DDLM_Set_Prm.req(Prm_Data: Unlock_Req=True)
```

WDATA **DEA4** **UNLCK**
DDLM_Data_Exchange.con
\Active=False
=> Input_Data=0
Diag.Station_Non_Existent=False
Continue_Slave_Handler.con(Diag=False)

WDATA **DS4** **DATA**
DDLM_Data_Exchange.con
\Active=True AND Status=DS
=> Continue_Slave_Handler.con(Diag=True)

WDATA **NA4** **DATA**
DDLM_Data_Exchange.con
\Active=True AND Status=NA
=> Diag.Station_Non_Existent=True
System_Diagnostic[x]=1
Continue_Slave_Handler.con(Diag=True)

WDATA **RE4** **DIAG1**
DDLM_Data_Exchange.con
\Active=True AND Status=RE/RS/RR/UE
=> Input_Data=0
Diag.Invalid_Slave_Response=True
Diag.Station_Non_Existent=False
System_Diagnostic[x]=1
Continue_Slave_Handler.con(Diag=True)

WDATA **OK4** **DATA**
DDLM_Data_Exchange.con(Inp_Data)
\Active=True AND Status=OK AND Diag_Flag=False
AND Inp_Data.len=exp_Inp_len
=> Input_Data=Inp_Data
Diag.Station_Non_Existent=False
Internal Data_Transfer_List[x]=1
Continue_Slave_Handler.con(Diag=False)

WDATA **ERR0** **UNLCK**
DDLM_Data_Exchange.con(Inp_Data)
\Active=True AND Status=OK AND Inp_Data.len # exp_Inp_len
AND (Inp_Data.len # 1 OR exp_Inp_len # 0
OR Diag_Flag = False)
=> Input_Data=0
Diag.Invalid_Slave_Response=True
Diag.Station_Non_Existent=False
System_Diagnostic[x]=1
Continue_Slave_Handler.con(Diag=True)

WDATA **DIAG1** **DIAG2**
DDLM_Data_Exchange.con(Inp_Data)
\Active=True AND Status=OK AND Diag_Flag=True AND
(Inp_Data.len=exp_Inp_len)
=> Input_Data=Inp_Data
Diag.Station_Non_Existent=False
Internal_Data_Transfer_List[x] = 1
Continue_Slave_Handler.con(Diag=False)

WDATA **DIAG2** **DIAG2**
DDLM_Data_Exchange.con(Inp_Data)
\Active=True AND Status=OK AND Diag_Flag=True AND
(Inp_Data.len=1 AND exp_Inp_len=0)
=> Diag.Station_Non_Existent=False
Internal_Data_Transfer_List[x] = 1
Continue_Slave_Handler.con(Diag=False)

WDATA **DEA5** **UNLCK**
DDLM_Set_Prm.con
\Active=False
=> Continue_Slave_Handler.con(Diag=False)

WDATA **DS5** **DATA**
DDLM_Set_Prm.con
\Active=True AND Status=DS
=> Continue_Slave_Handler.con(Diag=True)

WDATA **NA5** **DATA**
DDLM_Set_Prm.con
\Active=True AND Status=NA
=> Diag.Station_Non_Existent = True
System_Diagnostic[x] = 1
Continue_Slave_Handler.con(Diag=True)

WDATA **RE5** **DIAG1**
DDLM_Set_Prm.con
\Active=True AND Status=RE/RS/RR/UE
=> Diag.Invalid_Slave_Response = True
Continue_Slave_Handler.con(Diag=True)

WDATA **OK5** **DATA**
DDLM_Set_Prm.con
\Active=True AND Status=OK
=> New_Prm = False
Continue_Slave_Handler.con(Diag=True)

UNLCK **CONT8** **WUNLCK**
Continue_Slave_Handler.req(Output_Clear)
=> DDLM_Set_Prm.req(Prm_Data: Unlock_Req=True)

UNLCK **STOP6** **SUNLCK**
Stop_Slave_Handler.req
=> DDLM_Set_Prm.req(Prm_Data: Unlock_Req=True)

WUNLCK **UNLCK1** **DIAG1**
DDLM_Set_Prm.con
\Active=True
=> Continue_Slave_Handler.con(Diag=True)

WUNLCK **UNLCK2** **DEACT**
DDLM_Set_Prm.con
\Active=False
=> Diag.Deactivated=True
System_Diagnostic[x]=0
Continue_Slave_Handler.con(Diag=False)

SUNLCK	**STOP7**	**STOP**
DDLM_Set_Prm.con => Input_Data = 0 Diag.Deactivated=True Stop_Slave_Handler.con		
any-state	**FATAL1**	**STOP**
Reset_Slave_Handler.req		

13.3.2 State Machine of Scheduler

13.3.2.1 State Diagram of Scheduler

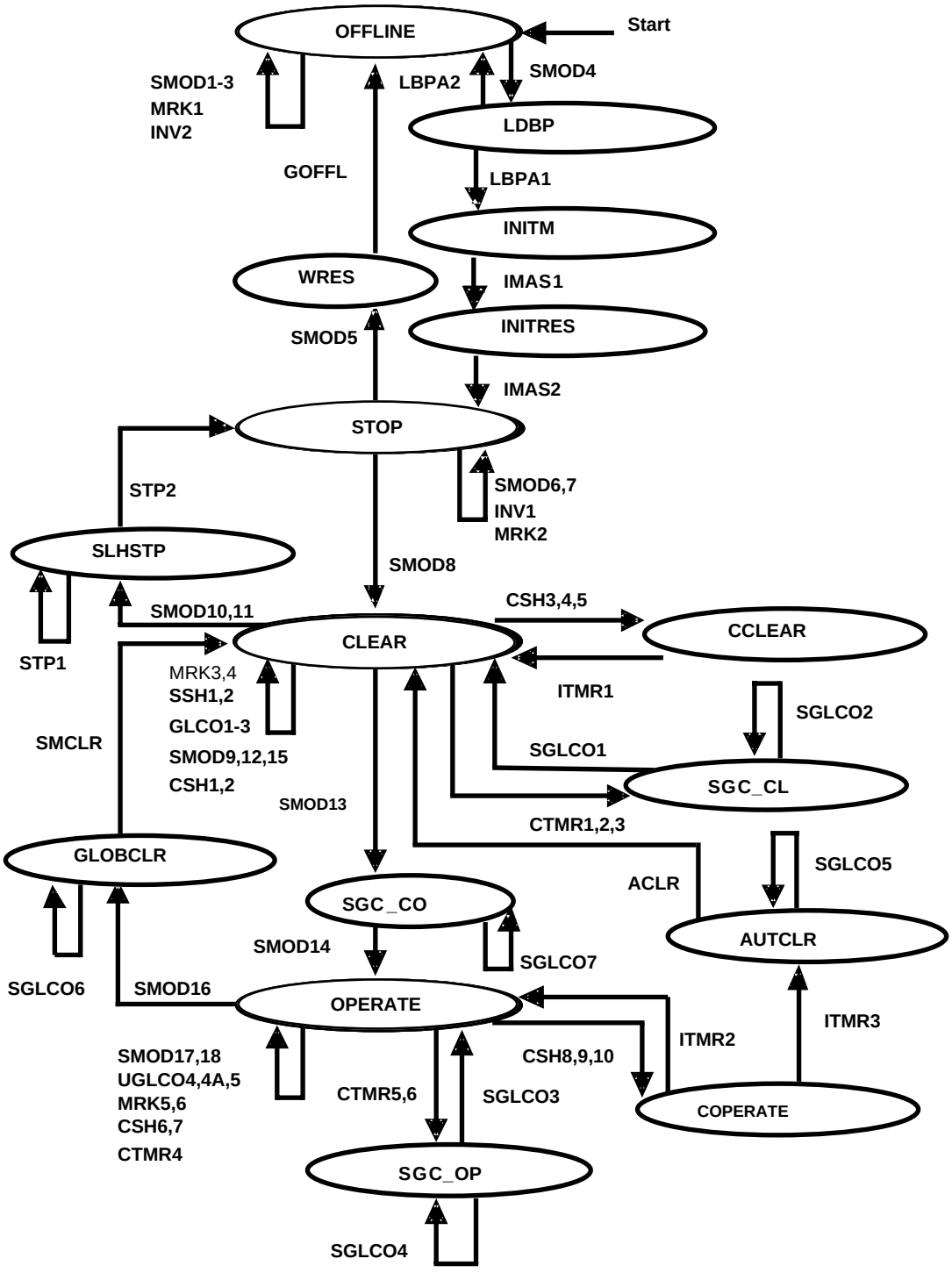


Figure 1. State Diagram of Scheduler at the DP-Master (class 1)

13.3.2.2 State Machine Description

The Scheduler performs the control of the individual Slave-Handlers. The Scheduler can be controlled by the user with the functions described in section "Service Interface". The Slave-Handlers are called, respectively stepped forward, one after the other from the Scheduler to compute the necessary actions of the Slave-Handlers. The Scheduler notifies the operation modes Clear and Operate in fixed time intervals (Dx_Control_Timer) to the DP-Slaves. This occures also at state transitions of the Scheduler.

The user and the Scheduler interact via the data interface using the following parameters:

Input_Data
(Octet-String)
Input data of the DP-Slaves, which are issued at the interface between the User-Interface and the user.

Data_Transfer_List
(Octet-String)
This list indicates, which DP-Slaves executed a data transfer cycle within the last control interval.

The following functions are called at the service interface:

- Set_Mode
- Mode_Changed
- Mark
- Global_Control

The description of these functions is given in section "Service Interface".

The local variables of the Scheduler are described subsequently:

Internal_Data_Transfer_List[0...125]
(Octet-String)
Contains the internal image of the Data_Transfer_List. The individual bits are set by the corresponding Slave-Handler and cleared by the Scheduler after copying them into the Data_Transfer_List.

GO_ACLR
(Boolean)
State variable which indicates that after finishing the current cycle the Scheduler shall branch to the state AUTOCLEAR.

Mark_Active
(Unsigned8)
Indicates that at present:
0 => local function Mark is not active
1 => local function Mark is active, but not yet in execution
2 => local function Mark is active and in execution

Diag_Active
(Boolean)
Used to store the status if during one pass trough the Slave_List a DP-Slave has not been in the user data exchange mode. The status which was stored is issued to the user with the Mark.con.

UGLCO_Active
(Boolean)
State variable which indicates that a local confirmation for a Global_Control function where expected, which where initiated by the user.

Additionally two timers are used:

- Min_Sl_Interval_Timer
  Supervises the minimal access control time of the DP-Slaves. This timer will be evaluated only in the states CCLEAR and COPEARTE.

- Dx_Control_Interval_Timer
  Supervises the correct execution of data transfer cycles. This timer will be evaluated only in the main states Clear and Operate.

13.3.2.3 State Transitions

Power_On **START** **OFFLINE**
=> Input_Data = 0
Diag = Initial Values
Internal_Data_Transfer_List[0..125] = 0
Data_Transfer_List[0..125] = 0
UGLCO_Active = False

OFFLINE **SMOD1** **OFFLINE**
Set_Mode.req(USIF_State)
\USIF_State # Stop AND USIF_State # Offline
=> Set_Mode.con(NO)
USIF_State=Offline

OFFLINE **SMOD2** **OFFLINE**
Set_Mode.req(USIF_State)
\USIF_State = Offline
=> Set_Mode.con(OK)

OFFLINE **SMOD3** **OFFLINE**
Set_Mode.req(USIF_State)
\USIF_State = Stop AND invalid Bus Parameter
=> Set_Mode.con(NO)

OFFLINE **SMOD4** **LDBP**
Set_Mode.req(USIF_State)
\USIF_State = Stop AND valid Bus Parameter
=> DDLM_Set_Bus_Par.req(Bus_Par)
Mark_Active = 0
Input_Data = 0
Diag_Active = False
Diag = Initial Values

OFFLINE **MRK1** **OFFLINE**
Mark.req
=> Mark.con(NO)

OFFLINE **INV2** **OFFLINE**
Global_Control.req(Rem_Add, Control_Command, Group_Select)
=> Global_Control.con(NO)

LDBP **LBPA1** **INITM**
DDLM_Set_Bus_Par.con(Status)
\Status=OK
=> DDLM_Master_Init.req

LDBP **LBPA2** **OFFLINE**
DDLM_Set_Bus_Par.con(Status)
\Status#OK
=> Set_Mode.con(NO)

INITM **IMAS1** **INITRES**
DDLM_Master_Init.con
=> DDLM_Responder_Init.req(Poll_Timeout)

INITRES **IMAS2** **STOP**
DDLM_Responder_Init.con
=> Set_Mode.con(OK)

STOP **SMOD5** **WRES**
Set_Mode.req(USIF_State)
\USIF_State = Offline
=> DDLM_Reset.Req

STOP **SMOD6** **STOP**
Set_Mode.req(USIF_State)
\USIF_State = Stop
=> Set_Mode.con(OK)

STOP **SMOD7** **STOP**
Set_Mode.req(USIF_State)
\USIF_State = Operate
=> Set_Mode.con(NO)

STOP **SMOD8** **CLEAR**
Set_Mode.req(USIF_State)
\USIF_State = Clear
=> Input_Data = 0
GO_ACLR = Error_Action_Flag
Internal_Data_Transfer_List[0..125] = 0
Start Dx_Control_Interval_Timer(Data_Control_Time/2)
Start_Slave_Handler.req(0 .. 125)

STOP **MRK2** **STOP**
Mark.req
=> Mark.con(NO)

STOP **INV1** **STOP**
Global_Control.req(Rem_Add, Control_Command, Group_Select)
=> Global_Control.con(NO)

```
CLEAR                              SMOD9                              CLEAR
    Set_Mode.req(USIF_State)
     \USIF_State = Offline
     =>  Set_Mode.con(NO)
CLEAR                              MRK3                               CLEAR
    Mark.req
     \Mark_Active = 0
     =>  Mark_Active = 1

CLEAR                              MRK4                               CLEAR
    Mark.req
     \Mark_Active # 0
     =>  Mark.con(NO)

CLEAR                              SMOD10                             SLHSTP
    Set_Mode.req(USIF_State)
     \USIF_State = Stop AND Mark_Active = 0
     =>  Data_Transfer_List[0..125] = 0
         Stop Dx_Control_Interval_Timer
         Stop_Slave_Handler.req(0..125)

CLEAR                              SMOD11                             SLHSTP
    Set_Mode.req(USIF_State)
     \USIF_State = Stop AND Mark_Active # 0
     =>  Mark.con(NO)
         Mark_Active = 0
         Data_Transfer_List[0..125] = 0
         Stop Dx_Control_Interval_Timer
         Stop_Slave_Handler.req(0..125)

SLHSTP                             STP1                               SLHSTP
    Stop_Slave_Handler.con(x)
     \not yet received all Stop_Slave_Handler.con
     =>  ignore


SLHSTP                             STP2                                STOP
    Stop_Slave_Handler.con(x)
     \received all Stop_Slave_Handler.con
     =>  Set_Mode.con(OK)

CLEAR                              SMOD12                             CLEAR
    Set_Mode.req(USIF_State)
     \USIF_State = Clear
     =>  Set_Mode.con(OK)

CLEAR                              SMOD13                             SGC_CO
    Set_Mode.req(USIF_State)
     \USIF_State = Operate AND GO_ACLR = False
     =>  DDLM_Global_Control.req(Rem_Add=127,
           Control_Command=0, Group_Select=0)

CLEAR                              SMOD15                             CLEAR
    Set_Mode.req(USIF_State)
     \USIF_State = Operate AND GO_ACLR = True
     =>  Set_Mode.con(NO)
```

SGC_CO **SGLCO7** **SGC_CO**
DDLM_Global_Control.con(Status)
\UGLCO_Active = True
=> UGLCO_Active = False
Global_Control.con(Status)

SGC_CO **SMOD14** **OPERATE**
DDLM_Global_Control.con(Status)
\UGLCO_Active = False
=> Set_Mode.con(OK)

CLEAR **GLCO1** **CLEAR**
Global_Control.req(Rem_Add, Control_Command, Group_Select)
\received all Start_Slave_Handler.con
=> UGLCO_Active = True
DDLM_Global_Control.req(Rem_Add,
Control_Command.1 = 1, Group_Select)

CLEAR **GLCO2** **CLEAR**
Global_Control.req(Rem_Add, Control_Command, Group_Select)
\not yet received all Start_Slave_Handler.con
=> Global_Control.con(NO)

CLEAR **GLCO3** **CLEAR**
DDLM_Global_Control.con(Status)
=> UGLCO_Active = False
Global_Control.con(Status)

CLEAR **SSH1** **CLEAR**
Start_Slave_Handler.con(x)
\not yet received all Start_Slave_Handler.con
=> ignore

CLEAR **SSH2** **CLEAR**
Start_Slave_Handler.con(x)
\received all Start_Slave_Handler.con
=> Start Min_Sl_Interval_Timer(Min_Slave_Interval)
Set_Mode.con(OK)
Continue_Slave_Handler.req(0..125,Output_Clear=True)

CLEAR **CSH1** **CLEAR**
Continue_Slave_Handler.con(x,Diag)
\not yet received all Continue_Slave_Handler.con
AND Mark_Active = 2
=> Diag_Active=Diag_Active OR Diag

CLEAR **CSH2** **CLEAR**
Continue_Slave_Handler.con(x,Diag)
\not yet received all Continue_Slave_Handler.con
AND Mark_Active # 2
=> ignore

CLEAR **CSH3** **CCLEAR**
Continue_Slave_Handler.con(x,Diag)
\received all Continue_Slave_Handler.con
AND Mark_Active = 0

```
CLEAR                                CSH4                              CCLEAR
    Continue_Slave_Handler.con(x,Diag)
     \received all Continue_Slave_Handler.con
      AND Mark_Active = 1
     =>  Mark_Active=2
         Diag_Active = False

CLEAR                                CSH5                              CCLEAR
    Continue_Slave_Handler.con(x,Diag)
     \received all Continue_Slave_Handler.con
      AND Mark_Active = 2
     =>  Diag_Active=Diag_Active OR Diag
         Mark_Active=0
         Mark.con(Status=OK, Dia=Diag_Active)

CLEAR                                CTMR1                             SGC_CL
    Dx_Control_Interval_Timer expired
     \Error_Action_Flag = True AND
      At least one Slave(i=0..125) fulfills following condition:
       Active[i] = True AND Internal_Data_Transfer_List[i] = 0
     =>  Data_Transfer_List[0..125]
                               = Internal_Data_Transfer_List[0..125]
         Internal_Data_Transfer_List[0..125] = 0
         Start Dx_Control_Interval_Timer(Data_Control_Time/2)
         GO_ACLR = True
           DDLM_Global_Control.req (Rem_Add = 127
           Control_Command=2, Group_Select = 0)

CLEAR                                CTMR2                             SGC_CL
    Dx_Control_Interval_Timer expired
     \Error_Action_Flag = True AND
      No Slave(i=0..125) fulfills following condition:
      Active[i] = True AND Internal_Data_Transfer_List[i] = 0
     =>  Data_Transfer_List[0..125]
                               = Internal_Data_Transfer_List[0..125]
         Internal_Data_Transfer_List[0..125] = 0
         GO_ACLR = False
         Start Dx_Control_Interval_Timer(Data_Control_Time/2)
           DDLM_Global_Control.req (Rem_Add = 127
           Control_Command=2, Group_Select = 0)

CLEAR                                CTMR3                             SGC_CL
    Dx_Control_Interval_Timer expired
     \Error_Action_Flag = False
     =>  Data_Transfer_List[0..125]
                               = Internal_Data_Transfer_List[0..125]
         Internal_Data_Transfer_List[0..125] = 0
         Start Dx_Control_Interval_Timer(Data_Control_Time/2)
           DDLM_Global_Control.req (Rem_Add = 127
           Control_Command=2, Group_Select = 0)

SGC_CL                               SGLCO1                            CLEAR
    DDLM_Global_Control.con(Status)
     \UGLCO_Active = False
    =>   ignore
```

SGC_CL **SGLCO2** **SGC_CL**

DDLM_Global_Control.con(Status)
\UGLCO_Active = True
=> UGLCO_Active = False
Global_Control.con(Status)

CCLEAR **ITMR1** **CLEAR**

Min_Sl_Interval_Timer expired
=> Continue_Slave_Handler.req(0..125,Output_Clear=True)
Start Min_Sl_Interval_Timer(Min_Slave_Interval)

OPERATE **SMOD17** **OPERATE**

Set_Mode.req(USIF_State)
\USIF_State # Clear AND USIF_State # Operate
=> Set_Mode.con(NO)

OPERATE **SMOD18** **OPERATE**

Set_Mode.req(USIF_State)
\USIF_State = Operate
=> Set_Mode.con(OK)

OPERATE **SMOD16** **GLOBCLR**

Set_Mode.req(USIF_State)
\USIF_State = Clear
=> DDLM_Global_Control.req(Rem_Add=127,
Control_Command=2, Group_Select=0)

OPERATE **MRK5** **OPERATE**

Mark.req
\Mark_Active = 0
=> Mark_Active = 1

OPERATE **MRK6** **OPERATE**

Mark.req
\Mark_Active # 0
=> Mark.con(NO)

OPERATE **UGLCO4** **OPERATE**

Global_Control.req(Rem_Add, Control_Command, Group_Select)
\Control_Command ≠ Clear
=> UGLCO_Active = True
DDLM_Global_Control.req(Rem_Add, Control_Command.1 = 0,
Group_Select)

OPERATE **UGLCO4A** **OPERATE**

Global_Control.req(Rem_Add, Control_Command, Group_Select)
\Control_Command = Clear
=> Global_Control.con(Status=NO)

OPERATE **UGLCO5** **OPERATE**

DDLM_Global_Control.con(Status)
=> UGLCO_Active = False
Global_Control.con(Status)

OPERATE **CSH6** **OPERATE**

Continue_Slave_Handler.con(x,Diag)
\not yet received all Continue_Slave_Handler.con
AND Mark_Active = 2
=> Diag_Active=Diag_Active OR Diag

OPERATE **CSH7** **OPERATE**

Continue_Slave_Handler.con(x,Diag)

```
 \not yet received all Continue_Slave_Handler.con
  AND Mark_Active # 2
 =>  ignore
```

OPERATE **CSH8** **COPERATE**

Continue_Slave_Handler.con(x,Diag)

```
 \received all Continue_Slave_Handler.con
  AND Mark_Active = 0
```

OPERATE **CSH9** **COPERATE**

Continue_Slave_Handler.con(x,Diag)

```
\received all Continue_Slave_Handler.con
  AND Mark_Active = 1
 =>  Mark_Active=2
     Diag_Active = False
```

OPERATE **CSH10** **COPERATE**

Continue_Slave_Handler.con(x,Diag)

```
 \received all Continue_Slave_Handler.con
  AND Mark_Active = 2
 =>  Diag_Active=Diag_Active OR Diag
     Mark_Active=0
     Mark.con(Status=OK, Dia=Diag_Active)
```

OPERATE **CTMR4** **OPERATE**

Dx_Control_Interval_Timer expired

```
 \Error_Action_Flag = True AND
  At least one Slave(i=0..125) fulfills following condition:
  Active[i] = True AND Internal_Data_Transfer_List[i] = 0
 =>  Data_Transfer_List[0..125]
                          = Internal_Data_Transfer_List[0..125]
     Internal_Data_Transfer_List[0..125] = 0
     Start Dx_Control_Interval_Timer(Data_Control_Time/2)
     GO_ACLR = True
```

OPERATE **CTMR5** **SGC_OP**

Dx_Control_Interval_Timer expired

```
 \Error_Action_Flag = True AND
  No Slave(i=0..125) fulfills following condition:
  Active[i] = True AND Internal_Data_Transfer_List[i] = 0
 =>  Data_Transfer_List[0..125]
                          = Internal_Data_Transfer_List[0..125]
     Internal_Data_Transfer_List[0..125] = 0
     Start Dx_Control_Interval_Timer(Data_Control_Time/2)
       DDLM_Global_Control.req (Rem_Add = 127
       Control_Command=0, Group_Select = 0)
```

```
OPERATE                              CTMR6                              SGC_OP
    Dx_Control_Interval_Timer expired
     \Error_Action_Flag = False
     =>  Data_Transfer_List[0..125]
                               = Internal_Data_Transfer_List[0..125]
         Internal_Data_Transfer_List[0..125] = 0
         Start Dx_Control_Interval_Timer(Data_Control_Time/2)
           DDLM_Global_Control.req (Rem_Add = 127
           Control_Command=0, Group_Select = 0)

SGC_OP                               SGLCO3                             OPERATE
    DDLM_Global_Control.con(Status)
     \UGLCO_Active = False
     =>  ignore

SGC_OP                               SGLCO4                             SGC_OP
    DDLM_Global_Control.con(Status)
     \UGLCO_Active = True
     =>  UGLCO_Active = False
         Global_Control.con(Status)


COPERATE                             ITMR2                              OPERATE
    Min_Sl_Interval_Timer expired
     \GO_ACLR = False
     =>  Continue_Slave_Handler.req(0..125,Output_Clear=False)
         Start Min_Sl_Interval_Timer(Min_Slave_Interval)

COPERATE                             ITMR3                              AUTCLR
    Min_Sl_Interval_Timer expired
     \GO_ACLR = True
     =>  DDLM_Global_Control.req (Rem_Add = 127
           Control_Command=2, Group_Select = 0)
         Start Min_Sl_Interval_Timer(Min_Slave_Interval)

AUTCLR                               SGLCO5                             AUTCLR
    DDLM_Global_Control.con(Status)
     \UGLCO_Active = True
     => UGLCO_Active = False
         Global_Control.con(Status)


AUTCLR                               ACLR                               CLEAR
    DDLM_Global_Control.con(Status)
     \UGLCO_Active = False
     =>  Mode_Changed.ind(USIF_State=Clear)
         Continue_Slave_Handler.req(0..125,Output_Clear=True)


GLOBCLR                              SGLCO6                             GLOBCLR
    DDLM_Global_Control.con(Status)
     \UGLCO_Active = True
     => UGLCO_Active = False
         Global_Control.con(Status)
```

GLOBCLR **SMCLR** **CLEAR**

DDLM_Global_Control.con(Status)

\UGLCO_Active = False

=> Set_Mode.con(OK)

any-state **FATAL1** **WRES**

DDLM_Fault.ind

=> Reset_Slave_Handler.req(0..125)

Mode_Changed.ind(USIF_State=Offline)

DDLM_Reset.req

WRES **GOFFL** **OFFLINE**

DDLM_Reset.con

13.3.3 State Machine of Service-Handler

13.3.3.1 State Diagram of Service-Handler

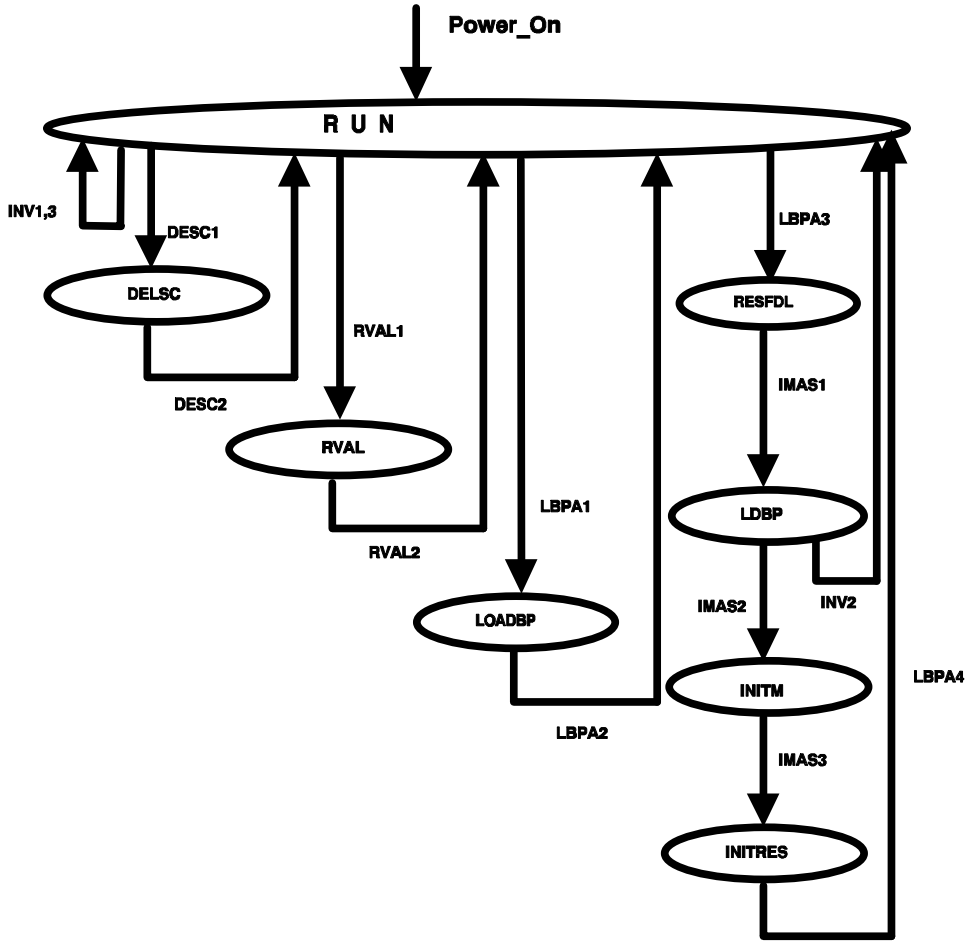


Figure 34. State Diagram of the Service-Handler at the DP-Master (class 1)

13.3.3.2 State Machine Description

The Service-Handler performs the local functions,

- Load_Bus_Par
- Read_Value
- Delete_SC

which are available for the user at the interface of the User-Interface of the DP-Master (class 1).
During the execution of the services, the state of the Scheduler (Scheduler_State) is read. At any time, only one service shall be active.

13.3.3.3 State Transitions

Power_On **START** **RUN**

RUN **INV1** **RUN**
Load_Bus_Par.req(Bus_Para)
\invalid Bus Parameter
=> Load_Bus_Par.con(IV)

RUN **LBPA1** **LOADBP**
Load_Bus_Par.req(Bus_Para)
\valid Bus Parameter AND critical Parameter unchanged
=> DDLM_Set_Bus_Par.req(Bus_Para)

LOADBP **LBPA2** **RUN**
DDLM_Set_Bus_Par.con(Status)
=> Load_Bus_Par.con(Status)

RUN **LBPA3** **RESFDL**
Load_Bus_Par.req(Bus_Para)
\valid Bus Parameter AND critical Parameter changed
AND (Scheduler_State = Offline OR Scheduler_State = Stop)
=> DDLM_Reset.req

RESFDL **IMAS1** **LDBP**
DDLM_Reset.con
=> DDLM_Set_Bus_Par.req(Bus_Para)

LDBP **IMAS2** **INITM**
DDLM_Set_Bus_Par.con(Status)
\Status=OK
=> DDLM_Master_Init.req

LDBP **INV2** **RUN**
DDLM_Set_Bus_Par.con(Status)
\Status#OK
=> Load_Bus_Para.con(Status)

INITM **IMAS3** **INITRES**
DDLM_Master_Init.con
=> DDLM_Responder_Init.req(Poll_Timeout)

INITRES **LBPA4** **RUN**
DDLM_Responder_Init.con
=> Load_Bus_Para.con(OK)

RUN **INV3** **RUN**
Load_Bus_Par.req(Bus_Para)
\valid Bus Parameter AND critical Parameter changed
AND (Scheduler_State # Offline AND Scheduler_State # Stop)
=> Load_Bus_Para.con(NO)

RUN **RVAL1** **RVAL**
Read_Value.req(Variable)
=> DDLM_Read_Value.req(Variable)

RVAL **RVAL2** **RUN**
DDLM_Read_Value.con(Status,Value)
=> Read_Value.con(Status,Value)

RUN **DESC1** **DELSC**
Delete_SC.req(Address)
=> DDLM_Delete_SC.req(Address)

DELSC **DESC2** **RUN**
DDLM_Delete_SC.con(Status)
=> Delete_SC.con(Status)

any-state **FATAL1** **RUN**
DDLM_Fault.ind

13.3.4 State Machine of DDLM

13.3.4.1 State Diagram of DDLM

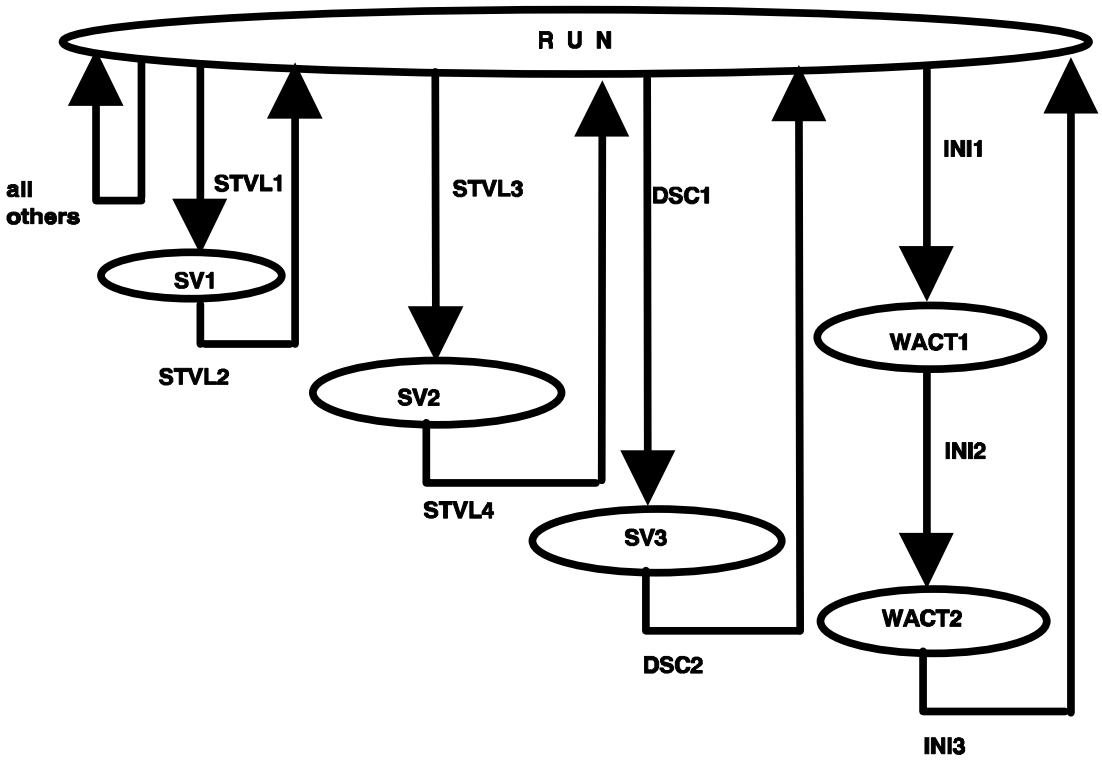


Figure 35. State Diagram of the DDLM of the DP-Master for DP-Slave communication.

13.3.4.2 State Machine Description

The following state machine specifies the behaviour of the DDLM of the DP-Master (class 1 and class 2) related to the slave handling. The DDLM parts for the master-master communication are specified in the next section. The individual DDLM parts of the master are operating independent from each other.

Basically, the DDLM of the DP-Master operates in the "RUN" state.

After power-on, the state machine changes to the "RUN" state.

This state machine assumes that the FDL and the FMA1/2 are able to accept the necessary number of parallel service requests. This parallelism is essentially determined by the number of existing slaves.

At the activation of the service access points (SAP), the maximum possible L_sdu length for the master-slave communication is taken from the Device data base.

The following local variables are used for the description of the state machine:

Max_Diag_Len
(Unsigned8)
Contains the maximal length of the diagnostic information which can be stored locally.

Loc_Station_Address
(Unsigned8)
Local variable for the intermediate storage of the own station address which was transferred with the last DDLM_Set_Bus_Par.

13.3.4.3 State Transitions

RUN RES1 RUN
DDLM_Reset.req
=> FMA1/2_RESET.req

RUN RES2 RUN
FMA1/2_RESET.con(OK)
=> ignore

RUN FATAL1 RUN
FMA1/2_RESET.con(NO/IV)
=> DDLM_Fault.ind

RUN INI1 WACT1
DDLM_Master_Init.req
=> FMA1/2_SAP_ACTIVATE.req(SSAP=NIL, Service_activate=SRD,
Role_in_service=Initiator)

WACT1 INI2 WACT2
FMA1/2_SAP_ACTIVATE.con(OK)
=> FMA1/2_SAP_ACTIVATE.req(SSAP=62,
Service_activate1=SRD, Role_in_service1=Initiator,
Service_activate2=SDN, Role_in_service2=Initiator)

WACT2 **INI3** **RUN**
FMA1/2_SAP_ACTIVATE.con(OK)
=> DDLM_Master_Init.con

any state **FATAL3** **RUN**
FMA1/2_SAP_ACTIVATE.con(IV/NO)
=> DDLM_Fault.ind

RUN **STVL1** **SV1**
DDLM_Set_Bus_Par.req(Bus_Para)
=> Loc_Station_Address = FDL_ADD
FMA1/2_SET_VALUE.req(
Variable_name1=TS,
Variable_name2=Baud_rate,
Variable_name3=T_{SL},
Variable_name4=min T_{SDR},
Variable_name5=max T_{SDR},
Variable_name6=T_{QUI},
Variable_name7=T_{SET},
Variable_name8=T_{TR},
Variable_name9=G,
Variable_name10=HSA,
Variable_name11=max_retry_limit,
Desired_Value1=FDL_Add
Desired_Value2=Baud_rate,
Desired_Value3=T_{SL},
Desired_Value4=min T_{SDR},
Desired_Value5=max T_{SDR},
Desired_Value6=T_{QUI},
Desired_Value7=T_{SET},
Desired_Value8=T_{TR},
Desired_Value9=G,
Desired_Value10=HSA,
Desired_Value11=max_retry_limit)

SV1 **STVL2** **RUN**
FMA1/2_SET_VALUE.con(Status)
=> DDLM_Set_Bus_Par.con(Status)

RUN **STVL3** **SV2**
DDLM_Set_Value.req(Variable,Value)
=> FMA1/2_SET_VALUE.req(
Variable_name1=Variable,
Desired_Value1=Value)

SV2 **STVL4** **RUN**
FMA1/2_SET_VALUE.con(Status)
=> DDLM_Set_Value.con(Status)

RUN **DSC1** **SV3**
DDLM_Delete_SC.req(Address)
=> FMA1/2_SET_VALUE.req(
Variable_name1=Frame_Sent_Count_(Address),
Variable_name2=Error_Count_(Address),
Desired_Value1=0,
Desired_Value2=0)

```
SV3                                   DSC2                                  RUN
    FMA1/2_SET_VALUE.con(Status)
     =>  DDLM_Delete_SC.con(Status)


RUN                                   DIAG1                                 RUN
    DDLM_Read_Value.req(Variable)
     =>  FMA1/2_READ_VALUE.req(Variable_name1=Variable)

RUN                                   RDVL1                                 RUN
    FMA1/2_READ_VALUE.con(Status,Value)
     =>  DDLM_Read_Value.con(Status,Value)

RUN                                   RDVL2                                 RUN
    DDLM_Slave_Diag.req(Rem_Add)
     =>  FDL_DATA_REPLY.req(SSAP=62, DSAP=60,
           Rem_add=Rem_Add, L_sdu.len=0, Serv_class=High)

RUN                                   DIAG2                                 RUN
    FDL_DATA_REPLY.con(DSAP=60)
     \L_status=DL AND L_sdu.len >= 6
      AND L_sdu.len <= Max_Diag_Len
      AND (L_sdu[4]=255 OR L_sdu[4]=Loc_Station_Address)
     =>  DDLM_Slave_Diag.con(Rem_Add=Rem_add, Status=OK,
           Diag_Data=L_sdu)


RUN                                   DIAG3                                 RUN
    FDL_DATA_REPLY.con(DSAP=60)
     \L_status=DL AND L_sdu.len >= 6
      AND L_sdu.len > Max_Diag_Len
      AND (L_sdu[4]=255 OR L_sdu[4]=Loc_Station_Address)
     =>  L_sdu[3].7=1         {Set Ext_Diag_Overflow}
         DDLM_Slave_Diag.con(Rem_Add=Rem_add, Status=OK,
           Diag_Data=L_sdu[1 - Max_Diag_Len])

RUN                                   DIAG4                                 RUN
    FDL_DATA_REPLY.con(DSAP=60)
     \L_status=DL AND L_sdu.len >= 6
      AND L_sdu.len > Max_Diag_Len
      AND (L_sdu[4]#255 AND L_sdu[4]#Loc_Station_Address)
     =>  L_sdu[1].7=1            {Set Master_Lock}
         L_sdu[3].7=1         {Set Ext_Diag_Overflow}
         DDLM_Slave_Diag.con(Rem_Add=Rem_add, Status=OK,
           Diag_Data=L_sdu[1 - Max_Diag_Len)


RUN                                   DIAG5                                 RUN
    FDL_DATA_REPLY.con(DSAP=60)
     \L_status=DL AND L_sdu.len >= 6
      AND L_sdu.len <= Max_Diag_Len
      AND (L_sdu[4]#255 AND L_sdu[4]#Loc_Station_Address)
     =>  L_sdu[1].7=1            {Set Master_Lock}
         DDLM_Slave_Diag.con(Rem_Add=Rem_add, Status=OK,
           Diag_Data=L_sdu)
```

```
RUN                                   DIAG6                                   RUN
    FDL_DATA_REPLY.con(DSAP=60)
     \L_status=DL AND L_sdu.len < 6
     =>  DDLM_Slave_Diag.con(Rem_Add=Rem_add, Status=RE)

RUN                                   DIAG7                                   RUN
    FDL_DATA_REPLY.con(DSAP=60)
     \L_status=LR/DH
     =>  DDLM_Slave_Diag.con(Rem_Add=Rem_add, Status=RE)


RUN                                   DIAG8                                   RUN
    FDL_DATA_REPLY.con(DSAP=60)
     \L_status=DS/UE/RS/NA/NR/RL/RDL/RDH
     =>  DDLM_Slave_Diag.con(Rem_Add=Rem_add, Status=L_status)

RUN                                   DATA1                                   RUN
    DDLM_Data_Exchange.req(Rem_Add, Outp_Data)
     =>  FDL_DATA_REPLY.req(SSAP=NIL, DSAP=NIL, Rem_add=Rem_Add,
           L_sdu=Outp_Data, Serv_class=high)

RUN                                   DATA2                                   RUN
    FDL_DATA_REPLY.con(DSAP=NIL)
     \L_status=NR
     =>  DDLM_Data_Exchange.con(Rem_Add=Rem_add, Status=OK,
           Diag_Flag=False, Inp_Data.len=0)


RUN                                   DATA3                                   RUN
    FDL_DATA_REPLY.con(DSAP=NIL)
     \L_status=DL
     =>  DDLM_Data_Exchange.con(Rem_Add=Rem_add, Status=OK,
           Diag_Flag=False, Inp_Data=L_sdu)

RUN                                   DATA4                                   RUN
    FDL_DATA_REPLY.con(DSAP=NIL)
     \L_status=DH
     =>  DDLM_Data_Exchange.con(Rem_Add=Rem_add, Status=OK,
           Diag_Flag=True, Inp_Data=L_sdu)

RUN                                   DATA5                                   RUN
    FDL_DATA_REPLY.con(DSAP=NIL)
     \L_status=DS/UE/RS/NA
     =>  DDLM_Data_Exchange.con(Rem_Add=Rem_add, Status=L_status)


RUN                                   DATA6                                   RUN
    FDL_DATA_REPLY.con(DSAP=NIL)
     \L_status=RR/RDL/RDH
     =>  DDLM_Data_Exchange.con(Rem_Add=Rem_add, Status=RR)

RUN                                   DATA7                                   RUN
    FDL_DATA_REPLY.con(DSAP=NIL)
     \L_status=LR
     =>  DDLM_Data_Exchange.con(Rem_Add=Rem_add, Status=RE)
```

```
RUN                                   INP1                                  RUN
    DDLM_RD_Inp.req(Rem_Add)
     =>  FDL_DATA_REPLY.req(SSAP=62, DSAP=56,
           Rem_add=Rem_Add, L_sdu.len=0, Serv_class=High)

RUN                                   INP2                                  RUN
    FDL_DATA_REPLY.con(DSAP=56)
     \L_status=DL
     =>  DDLM_RD_Inp.con(Rem_Add=Rem_add, Status=OK,
           Inp_Data=L_sdu)


RUN                                   INP3                                  RUN
    FDL_DATA_REPLY.con(DSAP=56)
     \L_status=DS/UE/RS/NA/NR/RL/RDL/RDH
     =>  DDLM_RD_Inp.con(Rem_Add=Rem_add, Status=L_status)


RUN                                   INP4                                  RUN
    FDL_DATA_REPLY.con(DSAP=56)
     \L_status=DH/LR
     =>  DDLM_RD_Inp.con(Rem_Add=Rem_add, Status=RE)

RUN                                   OUTP1                                 RUN
    DDLM_RD_Outp.req(Rem_Add)
     =>  FDL_DATA_REPLY.req(SSAP=62, DSAP=57,
           Rem_add=Rem_Add, L_sdu.len=0, Serv_class=High)

RUN                           OUTP2 RUN
      FDL_DATA_REPLY.con(DSAP=57)
            \L_status=DL
            =>    DDLM_RD_Outp.con(Rem_Add=Rem_add, Status=OK,
                        Outp_Data=L_sdu)


RUN                                   OUTP3                                 RUN
    FDL_DATA_REPLY.con(DSAP=57)
     \L_status=DS/UE/RS/NA/NR/RL/RDL/RDH
     =>  DDLM_RD_Outp.con(Rem_Add=Rem_add, Status=L_status)

RUN                                   OUTP4                                 RUN
    FDL_DATA_REPLY.con(DSAP=57)
     \L_status=DH/LR
     =>  DDLM_RD_Outp.con(Rem_Add=Rem_add, Status=RE)

RUN                                   SPRM1                                 RUN
    DDLM_Set_Prm.req(Rem_Add, Prm_Data)
     =>  FDL_DATA_REPLY.req(SSAP=62, DSAP=61,
           Rem_add=Rem_Add, L_sdu=Prm_Data, Serv_class=High)

RUN                                   SPRM2                                 RUN
    FDL_DATA_REPLY.con(DSAP=61)
     \L_status=NR
     =>  DDLM_Set_Prm.con(Rem_Add=Rem_add, Status=OK)

RUN                                   SPRM3                                 RUN
    FDL_DATA_REPLY.con(DSAP=61)
     \L_status=DS/NA/RS/RR/UE
     =>  DDLM_Set_Prm.con(Rem_Add=Rem_add, Status=L_status)
```

```
RUN                                   SPRM4                                  RUN
    FDL_DATA_REPLY.con(DSAP=61)
     \L_status=RDL/RDH/LR/DL/DH
     =>  DDLM_Set_Prm.con(Rem_Add=Rem_add, Status=RE)


RUN                                   SCFG1                                  RUN
    DDLM_Chk_Cfg.req(Rem_Add, Cfg_Data)
     =>  FDL_DATA_REPLY.req(SSAP=62, DSAP=62,
           Rem_add=Rem_Add, L_sdu=Cfg_Data, Serv_class=High)


RUN                                   SCFG2                                  RUN
    FDL_DATA_REPLY.con(DSAP=62)
     \L_status=NR
     =>  DDLM_Chk_Cfg.con(Rem_Add=Rem_add, Status=OK)

RUN                                   SCFG3                                  RUN
    FDL_DATA_REPLY.con(DSAP=62)
     \L_status=DS/NA/RS/RR/UE
     =>  DDLM_Chk_Cfg.con(Rem_Add=Rem_add, Status=L_status)

RUN                                   SCFG4                                  RUN
    FDL_DATA_REPLY.con(DSAP=62)
     \L_status=RDL/RDH/LR/DL/DH
     =>  DDLM_Chk_Cfg.con(Rem_Add=Rem_add, Status=RE)

RUN                                   GCFG1                                  RUN
    DDLM_Get_Cfg.req(Rem_Add)
     =>  FDL_DATA_REPLY.req(SSAP=62, DSAP=59,
           Rem_add=Rem_Add, L_sdu.len=0, Serv_class=High)

RUN                                   GCFG2                                  RUN
    FDL_DATA_REPLY.con(DSAP=59)
     \L_status=DL
     =>  DDLM_Get_Cfg.con(Rem_Add=Rem_add, Status=OK,
           Cfg_Data=L_sdu)


RUN                                   GCFG3                                  RUN
    FDL_DATA_REPLY.con(DSAP=59)
     \L_status=DS/UE/RS/NA/NR/RL/RDL/RDH
     =>  DDLM_Get_Cfg.con(Rem_Add=Rem_add, Status=L_status)

RUN                                   GCFG4                                  RUN
    FDL_DATA_REPLY.con(DSAP=59)
     \L_status=DH/LR
     =>  DDLM_Get_Cfg.con(Rem_Add=Rem_add, Status=RE)


RUN                                   GCTR1                                  RUN
    DDLM_Global_Control.req(Rem_Add, Control_Command,
        Group_Select)
     =>  FDL_DATA.req(SSAP=62, DSAP=58, Rem_add=Rem_Add,
           L_sdu[1]=Control_Command, L_sdu[2]=Group_Select,
           Serv_class=High)
```

```
RUN                                  GCTR2                                  RUN
    FDL_DATA.con(SSAP=62)
     \L_status=OK/DS
     =>  DDLM_Global_Control.con(Rem_Add=Rem_add, Status=L_status)

RUN                                  GCTR3                                  RUN
    FDL_DATA.con(SSAP=62)
     \L_status=LS/LR/IV
     =>  DDLM_Fault.ind

RUN                                  SADR1                                  RUN
    DDLM_Set_Slave_Address.req(Rem_Add, New_Slave_Add,
         Ident_Number, No_Add_Chg)
     =>  FDL_DATA_REPLY.req(SSAP=62, DSAP=55,
           Rem_add=Rem_Add, L_sdu[1]= New_Slave_Add,
           L_sdu[2-3]=Ident_Number, L_sdu[4]=No_Add_Chg)


RUN                                  SADR2                                  RUN
    DDLM_Set_Slave_Address.req(Rem_Add, New_Slave_Add,
         Ident_Number, No_Add_Chg, Rem_Slave_Data)
     =>  FDL_DATA_REPLY.req(SSAP=62, DSAP=55,
           Rem_add=Rem_Add, L_sdu[1]=New_Slave_Add,
           L_sdu[2-3]=Ident_Number, L_sdu[4]=No_Add_Chg,
           L_sdu[5-Rem_Slave_Data.len]=Rem_Slave_Data)


RUN                                  SADR3                                  RUN
    FDL_DATA_REPLY.con(DSAP=55)
     \L_status=NR
     =>  DDLM_Set_Slave_Address.con(Status=OK)

RUN                                  SADR4                                  RUN
    FDL_DATA_REPLY.con(DSAP=55)
     \L_status=DS/UE/RR/RS/NA
     =>  DDLM_Set_Slave_Address.con(Status=L_status)

RUN                                  SADR5                                  RUN
    FDL_DATA_REPLY.con(DSAP=55)
     \L_status=RDL/RDH/LR/DL/DH
     =>  DDLM_Set_Slave_Address.con(Status=RE)

RUN                                   ERR1                                  RUN
    FDL_DATA_REPLY.con(L_status=LS/IV)
     =>  DDLM_Fault.ind

RUN                                  ERR2                                   RUN
    unzulässige oder unbekannte FDL-primitive
     =>  DDLM_Fault.ind

RUN                                  ERR3                                   RUN
    FMA1/2_EVENT.ind(Event/Fault,Add_info)
     =>  DDLM_Event.ind(Event=Event/Fault,Add_Info=Add_info)
```

13.4 Communication Model of the DP-Master (Class 2)

13.4.1 General

The DP-Master (class 2) is subdivided in three instances:

1. Layer 1/2, Subset of PROFIBUS (FDL and FMA1/2)
2. Direct-Data-Link-Mapper (DDLM)
3. User

A summary of the interactions between the individual instances is shown in Figure 36.

13.4.2 FDL and FMA1/2 of PROFIBUS

Layer 2 for systems according to the PROFIBUS User Specifications (DP-Systems) shall realize at minimum the FDL-services SRD and SDN as initiator. Additionally, the Service SAP Activate FMA1/2 is prescribed as mandatory, besides the mandatory services Reset FMA1/2 and Event FMA1/2.
More hints can be taken from the sections "transmission technology" and "medium access and transmission protocol".

13.4.3 Direct-Data-Link-Mapper

The DDLM provides the mapping of the DDLM functions onto the layer 2 services.

The DDLM of the master (class 2) shall realize the following local functions as mandatory:

-DDLM_Master_Init
-DDLM_Requester_Init
-DDLM_Fault
-DDLM_Event

The following functions which can be called directly from the user of the DP-Master (class 2), are additionally possible:

-DDLM_Global_Control
-DDLM_RD_Inp
-DDLM_RD_Outp
-DDLM_Get_Cfg
-DDLM_Set_Slave_Add
-DDLM_Slave_Diag
-DDLM_Set_Prm
-DDLM_Chk_Cfg
-DDLM_Data_Exchange

The User-Interface handles the data exchange with the DP-Slaves with these functions.

The DDLM handles the communication between a DP-Master (class 2) and a DP-Master (class 1) transaction oriented. At any time, the DP-Master (class 1) can communicate with maximally one DP-Master (class 2). During this communication no parallel services are permitted.
The DP-Master (class 2) initiates all master-master communication functions. The DP-Master (class 1) does not need to answer these functions with an immediate response. In this case, the DP-Master (class 2) polls the DP-Master (class 1) cyclically until it offers the response data (see Figure 30).

The following master-master communication functions are offered optionally from the DDLM:

-DDLM_Get_Master_Diag
-DDLM_Start_Seq
-DDLM_End_Seq
-DDLM_Upload
-DDLM_Download
-DDLM_Act_Para_Brct
-DDLM_Act_Param

13.4.4 User

At the various devices, the user represents the real application process. It is defined application dependent. Indeed, it shall handle the communication with the DP-Slaves as described in section "State Machines in the DP-Master (Master-Slave)".

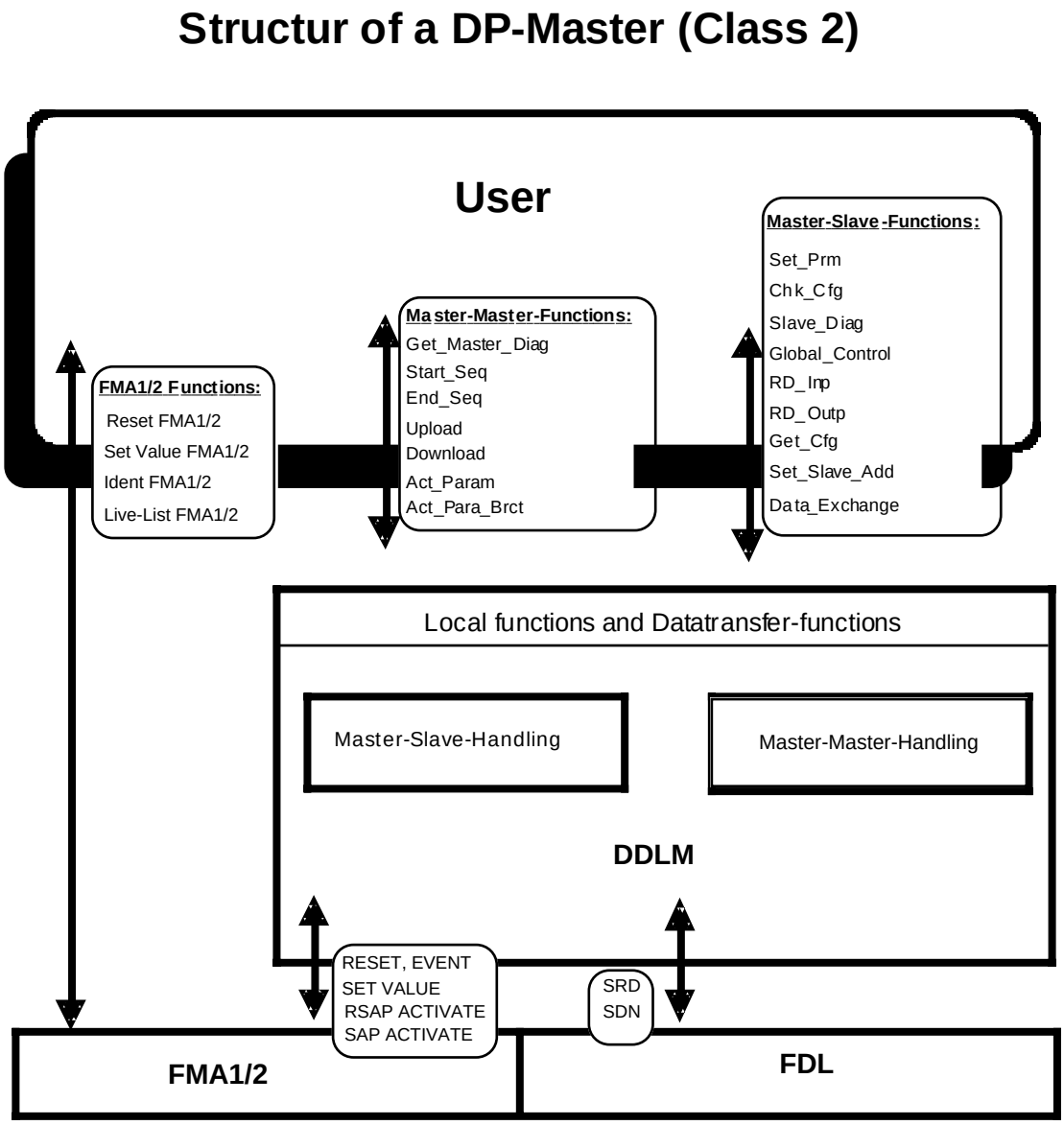


Figure 1. Structure of a DP-Master (class 2)

13.5 State Machines in the DP-Master (Master-Master)

13.5.1 State Machine of DDLM DP-Master (Class 1)

13.5.1.1 State Diagram of DDLM DP-Master (Class 1)

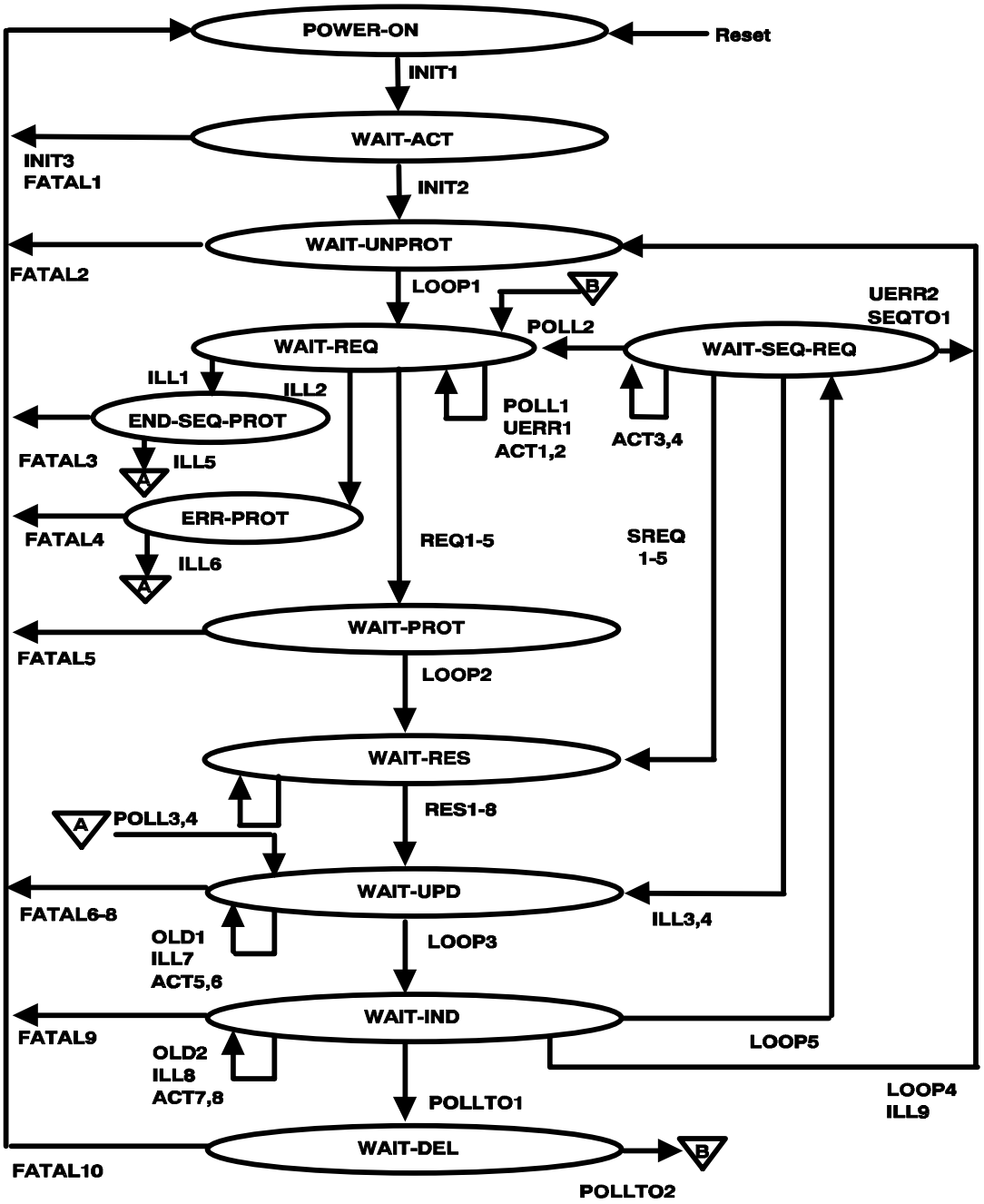


Figure 37. State Diagram for master-master communication at the DP-Master (class 1).

13.5.1.2 State Machine Description

The following DDLM state machine for master-master communication describes the responder side. Normally, this means a DP-interface of an automation device such as a Programmable Logic Controller.

After Reset, the DDLM changes to the "POWER-ON" state and waits for initialization by the user.

The state machine remains in the state "WAIT-REQ" as long as it receives a valid request from layer 2. After issuing this request to the user, the state machine remains in the "WAIT-RES" state until the user offers the response. After storing the response, data collection is monitored. If a service-sequence is executed, DDLM is waiting for the next service request in the state "WAIT-SEQ-REQ".

At any time, maximally one request shall be in execution.

At the activation of the service access points, the maximum possible L_sdu length for the master-master communication is taken from the Device data base.

Local variables of the DDLM:

T1
(Unsigned16)
Time in which a master (class 2) shall receive at latest a response from the master (class 1).

T2
(Unsigned16)
Time that can elapse during the execution of a service-sequence between response and the following request.

Client_Add
(Unsigned8)
Address of the DP-Master (class 2) just communicating with.

Service-Header
(Octet-String of the length 4)
Intermediate memory for the first four octets of the last request.

Sequence_Mode
(Boolean)
Indicates that a service-sequence is just in execution. During a service-sequence the DP-Master (class 1) is reserved for the respective DP-Master (class 2).

13.5.1.3 State Transitions

DDLM State Machine for Master-Master Communication
(Responder Side - DP-Master)

POWER-ON **INIT1** **WAIT-ACT**
DDLM_Responder_Init.req(Poll_Timeout)
=> FMA1/2_SAP_ACTIVATE.req(SSAP=54, Access=All,
Service_activate1=SDN, Role_in_service1=Responder)
Client_Add=Invalid
Sequence_Mode=False
T1=Poll_Timeout
Start T1

WAIT-ACT **INIT2** **WAIT-UNPROT**
FMA1/2_SAP_ACTIVATE.con(SSAP=54)
\M_status=OK
=> FMA1/2_RSAP_ACTIVATE.req(SSAP=54, Access=All,
Indication_Mode=Data)
DDLM_Responder_Init.con

WAIT-ACT **INIT3** **POWER-ON**
FMA1/2_SAP_ACTIVATE.con(SSAP=54)
\M_status=NO/IV
=> DDLM_Fault.ind

WAIT-ACT **FATAL1** **POWER-ON**
T1 expired
=> DDLM_Fault.ind

WAIT-UNPROT **LOOP1** **WAIT-REQ**
FMA1/2_RSAP_ACTIVATE.con(SSAP=54)
\M_status = OK
=> Stop T1

WAIT-UNPROT **FATAL2** **POWER-ON**
T1 expired
=> DDLM_Fault.ind

WAIT-REQ **POLL1** **WAIT-REQ**
FDL_DATA_REPLY.ind(DSAP=54)
\L_sdu.len=0
=> ignore

WAIT-REQ **UERR1** **WAIT-REQ**
DDLM_xxx.res
=> ignore

WAIT-REQ **REQ1** **WAIT-PROT**
FDL_DATA_REPLY.ind(DSAP=54)
\L_sdu.len=2 AND L_sdu[1]=41H
=> DDLM_Get_Master_Diag.ind(Identifier=L_sdu[2])
Service_Header=L_sdu[1-2]
FMA1/2_RSAP_ACTIVATE.req(SSAP=54, Acces=Loc_add,
Indication_Mode=Unchange)
Client_Add=Loc_add
Start T1

WAIT-REQ **REQ2** **WAIT-PROT**

FDL_DATA_REPLY.ind(DSAP=54)

```
 \L_sdu.len=4 AND L_sdu[1]=42H
 =>  Sequence_Mode=True
     Service_Header=L_sdu[1-4]
     T2=L_sdu[3-4]
     DDLM_Start_Seq.ind(Req_Add=Loc_add, Area_Code=L_sdu[2],
       Timeout=L_sdu[3-4])
     FMA1/2_RSAP_ACTIVATE.req(SSAP=54, Access=Loc_add,
       Indication_Mode=Unchange)
     Client_Add=Loc_add
     Start T1
```

WAIT-REQ **REQ3** **WAIT-PROT**

FDL_DATA_REPLY.ind(DSAP=54)

```
 \L_sdu.len>=5 AND L_sdu[1]=43H
 =>  DDLM_Download.ind(Req_Add=Loc_add, Area_Code=L_sdu[2],
       Address_Offset=L_sdu[3-4], Data=L_sdu[5-L_sdu.len])
     Service_Header=L_sdu[1-4]
     FMA1/2_RSAP_ACTIVATE.req(SSAP=54, Access=Loc_add,
       Indication_Mode=Unchange)
     Client_Add=Loc_add
     Start T1
```

WAIT-REQ **REQ4** **WAIT-PROT**

FDL_DATA_REPLY.ind(DSAP=54)

```
 \L_sdu.len=5 AND L_sdu[1]=44H
 =>  DDLM_Upload.ind(Req_Add=Loc_add, Area_Code=L_sdu[2],
       Address_Offset=L_sdu[3-4], Data_Length=L_sdu[5])
     Service_Header=L_sdu[1-4]
     FMA1/2_RSAP_ACTIVATE.req(SSAP=54, Access=Loc_add,
       Indication_Mode=Unchange)
     Client_Add=Loc_add
     Start T1
```

WAIT-REQ **ILL1** **END-SEQ-PROT**

FDL_DATA_REPLY.ind(DSAP=54)

```
 \L_sdu.len=1 AND L_sdu[1]=45H  {End_Seq.req PDU}
 =>  FMA1/2_RSAP_ACTIVATE.req(SSAP=54, Access=Loc_add,
       Indication_Mode=Unchange)
     Client_Add=Loc_add
     Start T1
```

WAIT-REQ **REQ5** **WAIT-PROT**

FDL_DATA_REPLY.ind(DSAP=54)

```
 \L_sdu.len=3 AND L_sdu[1]=47H
 =>  DDLM_Act_Param.ind(Area_Code=L_sdu[2],
       Activate=L_sdu[3])
     Service_Header=L_sdu[1-3]
     FMA1/2_RSAP_ACTIVATE.req(SSAP=54, Access=Loc_add,
       Indication_Mode=Unchange     )
     Client_Add=Loc_add
     Start T1
```

WAIT-REQ **ILL2** **ERR-PROT**
FDL_DATA_REPLY.ind(DSAP=54)
\invalid DDLM-PDU
=> FMA1/2_RSAP_ACTIVATE.req(SSAP=54, Access=Loc_add,
Indication_Mode=Unchange)
Client_Add=Loc_add
Start T1

WAIT-REQ **ACT1** **WAIT-REQ**
FDL_DATA.ind(DSAP=54)
\L_sdu.len=2 AND L_sdu[1]=46H
=> DDLM_Act_Para_Brct.ind(Area_Code=L_sdu[2])

WAIT-REQ **ACT2** **WAIT-REQ**
FDL_DATA.ind(DSAP=54)
\L_sdu.len#2 OR L_sdu[1]#46H
=> ignore

WAIT-SEQ-REQ **POLL2** **WAIT-REQ**
FDL_DATA_REPLY.ind(DSAP=54)
\L_sdu.len=0
=> ignore

WAIT-SEQ-REQ **UERR2** **WAIT-UNPROT**
DDLM_xxx.res
=> Sequence_Mode=False
FMA1/2_RSAP_ACTIVATE.req(SSAP=54, Access=All,
Indication_Mode=Unchange)
Stop T2

WAIT-SEQ-REQ **SREQ1** **WAIT-RES**
FDL_DATA_REPLY.ind(DSAP=54)
\L_sdu.len=2 AND L_sdu[1]=41H
=> DDLM_Get_Master_Diag.ind(Identifier=L_sdu[2])
Service_Header=L_sdu[1-2]
Stop T2
Start T1

WAIT-SEQ-REQ **ILL3** **WAIT-UPD**
FDL_DATA_REPLY.ind(DSAP=54)
\L_sdu.len=3 AND L_sdu[1]=42H
=> Sequence_Mode=False
FDL_REPLY_UPDATE.req(SSAP=54, L_sdu[1]=C9H,
Serv_class=Low, Transmit=Single) {SE error code}
Stop T2
Start T1

WAIT-SEQ-REQ **SREQ2** **WAIT-RES**

FDL_DATA_REPLY.ind(DSAP=54)

```
 \L_sdu.len>=5 AND L_sdu[1]=43H
 => DDLM_Download.ind(Req_Add=Loc_add, Area_Code=L_sdu[2],
      Address_Offset=L_sdu[3-4], Data=L_sdu[5-L_sdu.len])
    Service_Header=L_sdu[1-4]
    Stop T2
    Start T1
```

WAIT-SEQ-REQ **SREQ3** **WAIT-RES**

FDL_DATA_REPLY.ind(DSAP=54)

```
 \L_sdu.len=5 AND L_sdu[1]=44H
 => DDLM_Upload.ind(Req_Add=Loc_add, Area_Code=L_sdu[2],
      Address_Offset=L_sdu[3-4], Data_Length=L_sdu[5])
    Service_Header=L_sdu[1-4]
    Stop T2
    Start T1
```

WAIT-SEQ-REQ **SREQ4** **WAIT-RES**

FDL_DATA_REPLY.ind(DSAP=54)

```
 \L_sdu.len=1 AND L_sdu[1]=45H
 => Sequence_Mode=False
    DDLM_End_Seq.ind(Req_Add=Loc_add)
    Service_Header=L_sdu[1]
    Stop T2
    Start T1
```

WAIT-SEQ-REQ **SREQ5** **WAIT-RES**

FDL_DATA_REPLY.ind(DSAP=54)

```
 \L_sdu.len=3 AND L_sdu[1]=47H
 => DDLM_Act_Param.ind(Area_Code=L_sdu[2],
      Activate=L_sdu[3])
    Service_Header=L_sdu[1-3]
    Stop T2
    Start T1
```

WAIT-SEQ-REQ **ILL4** **WAIT-UPD**

FDL_DATA_REPLY.ind(DSAP=54)

```
 \invalid DDLM-PDU AND Sequence_Mode=True
 => Sequence_Mode=False
    FDL_REPLY_UPDATE.req(SSAP=54, L_sdu[1]=C1H,
      Serv_class=Low, Transmit=Single)  {FE error code}
    Stop T2
    Start T1
```

WAIT-SEQ-REQ **SEQTO1** **WAIT-UNPROT**

T2 expired

```
 => Sequence_Mode=False
    FMA1/2_RSAP_ACTIVATE.req(SSAP=54, Access=All,
      Indication_Mode=Unchange)
```

WAIT-SEQ-REQ **ACT3** **WAIT-SEQ-REQ**
FDL_DATA.ind(DSAP=54)
\L_sdu.len=2 AND Loc_add=Client_Add AND L_sdu[1]=46H
=> DDLM_Act_Para_Brct.ind(Area_Code=L_sdu[2])

WAIT-SEQ-REQ **ACT4** **WAIT-SEQ-REQ**
FDL_DATA.ind(DSAP=54)
\L_sdu.len#2 OR Loc_add#Client_Add OR L_sdu[1]#46H
=> ignore

END-SEQ-PROT **ILL5** **WAIT-UPD**
FMA1/2_RSAP_ACTIVATE.con(SSAP=54)
\M_status=OK
=> FDL_REPLY_UPDATE.req(SSAP=54, L_sdu[1]=C9H,
Serv_class=Low, Transmit=Single) {SE error code}

END-SEQ-PROT **FATAL3** **POWER-ON**
T1 expired
=> DDLM_Fault.ind

ERR-PROT **ILL6** **WAIT-UPD**
FMA1/2_RSAP_ACTIVATE.con(SSAP=54)
\M_status=OK
=> FDL_REPLY_UPDATE.req(SSAP=54, L_sdu[1]=C1H,
Serv_class=Low, Transmit=Single) {FE error code}

ERR-PROT **FATAL4** **POWER-ON**
T1 expired
=> DDLM_Fault.ind

WAIT-PROT **LOOP2** **WAIT-RES**
FMA1/2_RSAP_ACTIVATE.con(OK)
=> ignore

WAIT-PROT **FATAL5** **POWER-ON**
T1 expired
=> DDLM_Fault.ind

WAIT-RES **POLL3** **WAIT-RES**
FDL_DATA_REPLY.ind(DSAP=54)
\Sequence_Mode=False OR Client_Add#Loc_add OR L_sdu.len=0
=> ignore

WAIT-RES **POLL4** **WAIT-RES**
FDL_DATA_REPLY.ind(DSAP=54)
\Sequence_Mode=True AND Client_Add=Loc_add AND L_sdu.len>0
=> Sequence_Mode=False
ignore L_sdu

WAIT-RES **RES1** **WAIT-UPD**
DDLM_Get_Master_Diag.res(Status, Diagnostic_Data)
\Service_Header[1]=41H AND Status=OK
=> FDL_REPLY_UPDATE.req(SSAP=54, L_sdu=(Service_Header,
Start T1

WAIT-RES **RES2** **WAIT-UPD**
DDLM_Start_Seq.res(Status, Max_Length_Data_Unit)
\Service_Header[1]=42H AND Status=OK
=> FDL_REPLY_UPDATE.req(SSAP=54, L_sdu=(Service_Header,
Start T1

WAIT-RES **RES3** **WAIT-UPD**
DDLM_Download.res(Status)
\Service_Header[1]=43H AND Status=OK
=> FDL_REPLY_UPDATE.req(SSAP=54, L_sdu=Service_Header,
Serv_class=Low,Transmit=Single)
Start T1

WAIT-RES **RES4** **WAIT-UPD**
DDLM_Upload.res(Status, Data)
\Service_Header[1]=44H AND Status=OK
=> FDL_REPLY_UPDATE.req(SSAP=54, L_sdu=(Service_Header,
Data), Serv_class=Low,Transmit=Single)
Start T1

WAIT-RES **RES5** **WAIT-UPD**
DDLM_End_Seq.res(Status)
\Service_Header[1]=45H AND Status=OK
=> FDL_REPLY_UPDATE.req(SSAP=54, L_sdu=Service_Header,
Serv_class=Low,Transmit=Single)
Start T1

WAIT-RES **RES6** **WAIT-UPD**
DDLM_Act_Param.res(Status)
\Service_Header[1]=47H AND Status=OK
=> FDL_REPLY_UPDATE.req(SSAP=54, L_sdu=Service_Header,
Serv_class=Low,Transmit=Single)
Start T1

WAIT-RES **RES7** **WAIT-UPD**
DDLM_xxx.res(Status)
\Service_Header[1]=Service_Code AND Status # OK
=> FDL_REPLY_UPDATE.req(SSAP=54, L_sdu[1]=Status,
Serv_class=Low, Transmit=Single)
Start T1

WAIT-RES **RES8** **WAIT-UPD**
DDLM_xxx.res(Status, ...)
\Service_Header[1]#Service_Code
=> FDL_REPLY_UPDATE.req(SSAP=54, L_sdu[1]=C6H,
Serv_class=Low, Transmit=Single) {RE}
Sequence_Mode=False

WAIT-UPD **LOOP3** **WAIT-IND**
FDL_REPLY_UPDATE.con(SSAP=54)
\Status=OK
=> ignore

WAIT-UPD **OLD1** **WAIT-UPD**
FDL_DATA_REPLY.ind(DSAP=54)
\Loc_add#Client_Add AND Update_status=NO
=> ignore

WAIT-UPD **ILL7** **WAIT-UPD**
FDL_DATA_REPLY.ind(DSAP=54)
\Loc_add=Client_Add AND L_sdu.len>0 AND Update_status=NO
=> Sequence_Mode = False
ignore L_sdu

WAIT-UPD **FATAL6** **POWER-ON**
FDL_DATA_REPLY.ind(DSAP=54)
\Update_status=LO/HI
=> DDLM_Fault.ind

WAIT-UPD **FATAL7** **POWER-ON**
FDL_REPLY_UPDATE.con(SSAP=54)
\Status=NO/IV
=> DDLM_Fault.ind

WAIT-UPD **FATAL8** **POWER-ON**
T1 expired
=> DDLM_Fault.ind

WAIT-UPD **ACT5** **WAIT-UPD**
FDL_DATA.ind(DSAP=54)
\L_sdu.len=2 AND L_sdu[1]=46H
AND (Sequence_Mode=False OR Loc_add=Client_Add)
=> DDLM_Act_Para_Brct.ind(Area_Code=L_sdu[2])

WAIT-UPD **ACT6** **WAIT-UPD**
FDL_DATA.ind(DSAP=54)
\L_sdu.len#2 OR L_sdu[1]#46H OR Sequence_Mode=True
OR Loc_add#Client_Add
=> ignore

WAIT-IND **OLD2** **WAIT-IND**
FDL_DATA_REPLY.ind(DSAP=54)
\Loc_add#Client_Add AND Update_status=NO
=> ignore

WAIT-IND **FATAL9** **POWER-ON**
FDL_DATA_REPLY.ind(DSAP=54)
\Update_status=LO/HI AND Loc_add#Client_Add
=> DDLM_Fault.ind

WAIT-IND **ILL8** **WAIT-IND**
FDL_DATA_REPLY.ind(DSAP=54)
\Loc_add=Client_Add AND L_sdu.len>0 AND Update_status=NO
=> Sequence_Mode = False
ignore L_sdu

WAIT-IND **LOOP4** **WAIT-UNPROT**
FDL_DATA_REPLY.ind(DSAP=54)
\Update_status=LO AND Sequence_Mode=False
=> FMA1/2_RSAP_ACTIVATE.req(SSAP=54, Access=All,
Indication_Mode=Unchange)
ignore L_sdu, if existend
Start T1

WAIT-IND **LOOP5** **WAIT-SEQ-REQ**
FDL_DATA_REPLY.ind(DSAP=54)
\Update_status=LO AND Sequence_Mode=True AND L_sdu.len=0
=> Stop T1
Start T2

WAIT-IND **ILL9** **WAIT-UNPROT**
FDL_DATA_REPLY.ind(DSAP=54)
\Update_status=LO AND Sequence_Mode=True AND L_sdu.len>0
=> FMA1/2_RSAP_ACTIVATE.req(SSAP=54, Access=All,
Indication_Mode=Unchange)
Sequence_Mode=False
ignore L_sdu, if existend
Start T1

WAIT-IND **ACT7** **WAIT-IND**
FDL_DATA.ind(DSAP=54)
\L_sdu.len=2 AND L_sdu[1]=46H
AND (Sequence_Mode=False OR Loc_add=Client_Add)
=> DDLM_Act_Para_Brct.ind(Area_Code=L_sdu[2])

WAIT-IND **ACT8** **WAIT-IND**
FDL_DATA.ind(DSAP=54)
\L_sdu.len#2 OR L_sdu[1]#46H
OR (Sequence_Mode=True AND Loc_add#Client_Add)
=> ignore

WAIT-IND **POLLTO1** **WAIT-DEL**
T1 expired
=> FDL_REPLY_UPDATE.req(SSAP=54, L_sdu.len=0,
Serv_class=Low, Transmit=Single)
Sequence_Mode=False
Start T1

WAIT-DEL **POLLTO2** **WAIT-REQ**
FDL_REPLY_UPDATE.con(SSAP=54)
\L_status=OK
=> FMA1/2_RSAP_ACTIVATE.req(SSAP=54, Access=All,
Indication_Mode=Unchange)

WAIT-DEL **FATAL10** **POWER-ON**
T1 expired
=> DDLM_Fault.ind

any-state	**FATAL11**	**POWER-ON**

FMA1/2_RSAP_ACTIVATE.con(SSAP=54)
\L_status=NO/IV
=> DDLM_Fault.ind

13.5.2 State Machine of DDLM DP-Master (Class 2)

13.5.2.1 State Diagram of DDLM DP-Master (Class 2)

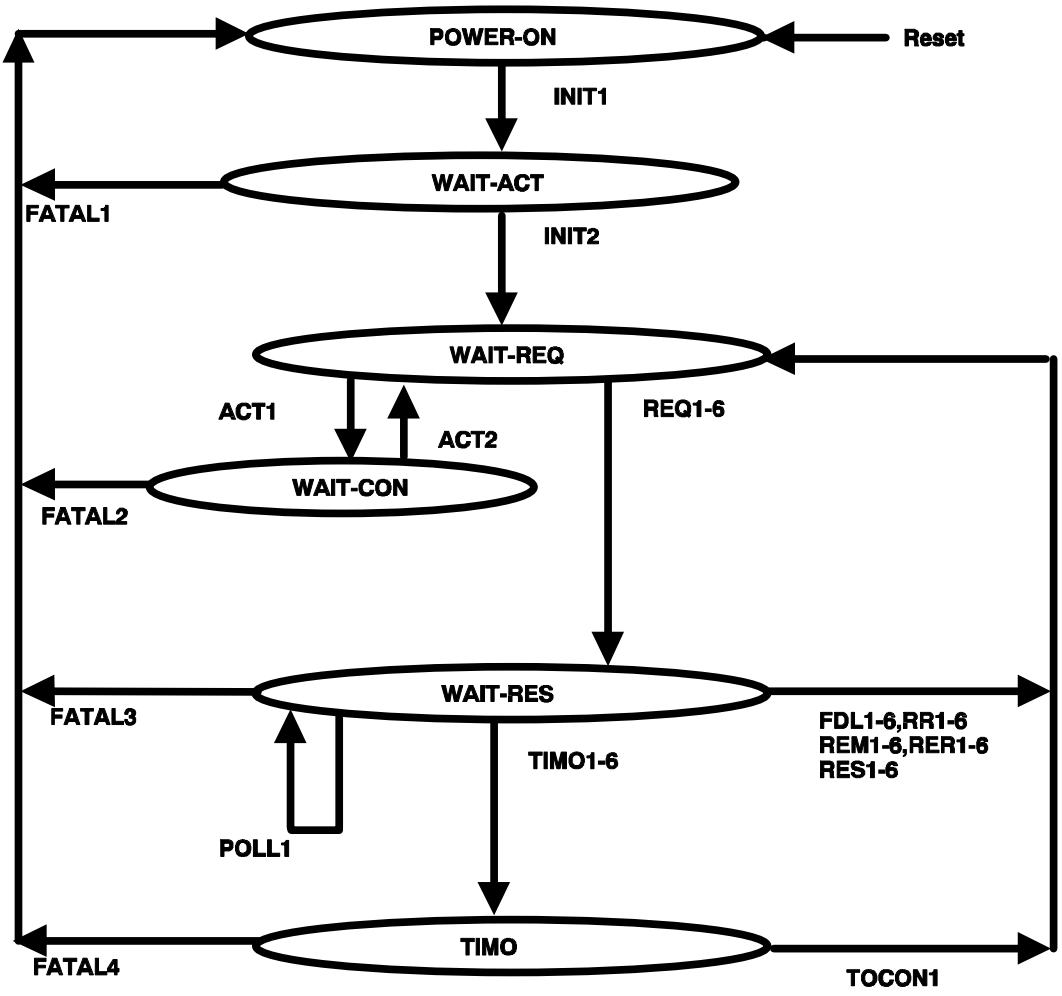


Figure 38. State Diagram for Master-Master communication of the DP-Master (class 2)

13.5.2.2 State Machine Description

The following DDLM state machine for master-master communication specifies the requester side. Typically, these are programming or diagnostic devices.

After reset, the DDLM changes to the "POWER-ON" state and waits for initialization by the user.

The state machine resumes as long in the "WAIT-REQ" state as it receives a valid request from the user. After this request was sent, the state machine remains in the "WAIT-RES" state, until the service is terminated successfully or incorrectly.

At any time, maximally one service request is in execution.

At the activation of the service access points, the maximum possible L_sdu length for the master-master communication is taken from the Device data base.

Local variables of the DDLM:

T1
(Unsigned16)
Time which can elapse at most between an indication and the receipt of the associated response from the user.

Server_Add
(Unsigned8)
Address of the DP-Master (class 1) just communicating with.

Service-Header
(Octet-String of length 4)
Intermediate memory for the first four octets of the last request.

13.5.2.3 State Transitions

DDLM State Machine for Master-Master Communication
(Requester Side - DP-Master)

POWER-ON **INIT1** **WAIT-ACT**
DDLM_Requester_Init.req(Poll_Timeout)
=> FMA1/2_SAP_ACTIVATE.req(SSAP=54, Service_activate1=SRD,
Role_in_service1=Requester)
T1=Poll_Timeout

WAIT-ACT **INIT2** **WAIT-REQ**
FMA1/2_SAP_ACTIVATE.con(SSAP=54)
\M_status=OK

WAIT-ACT **FATAL1** **POWER-ON**
FMA1/2_SAP_ACTIVATE.con(SSAP=54)
\M_status=NO/IV
=> DDLM_Fault.ind

WAIT-REQ **REQ1** **WAIT-RES**
DDLM_Get_Master_Diag.req(Rem_Add,Identifier)
=> Service_Header[1]=41H
Service_Header[2]=Identifier
FDL_DATA_REPLY.req(SSAP=54, DSAP=54, Rem_add=Rem_Add,
L_sdu=Service_Header, Serv_class=Low)
Server_Add=Rem_Add
Start T1

WAIT-REQ **REQ2** **WAIT-RES**
DDLM_Start_Seq.req(Rem_Add,Area_Code,Timeout)
=> Service_Header[1]=42H
Service_Header[2]=Area_Code
Service_Header[3-4]=Timeout
FDL_DATA_REPLY.req(SSAP=54, DSAP=54, Rem_add=Rem_Add,
L_sdu=Service_Header, Serv_class=Low)
Server_Add=Rem_Add
Start T1

WAIT-REQ **REQ3** **WAIT-RES**
DDLM_Download.req(Rem_Add,Area_Code,Address_Offset,Data)
=> Service_Header[1]=43H
Service_Header[2]=Area_Code
Service_Header[3-4]=Address_Offset
FDL_DATA_REPLY.req(SSAP=54, DSAP=54, Rem_add=Rem_Add,
L_sdu=(Service_Header,Data), Serv_class=Low)
Server_Add=Rem_Add
Start T1

WAIT-REQ **REQ4** **WAIT-RES**
DDLM_Upload.req(Rem_Add,Area_Code,Address_Offset,
Data_Length)
=> Service_Header[1]=44H
Service_Header[2]=Area_Code
Service_Header[3-4]=Address_Offset
Service_Header[5]=Data_Length
FDL_DATA_REPLY.req(SSAP=54, DSAP=54, Rem_add=Rem_Add,
L_sdu=(Service_Header,Data_Length), Serv_class=Low)
Server_Add=Rem_Add
Start T1

WAIT-REQ **REQ5** **WAIT-RES**
DDLM_End_Seq.req(Rem_Add
=> Service_Header[1]=45H
FDL_DATA_REPLY.req(SSAP=54, DSAP=54, Rem_add=Rem_Add,
L_sdu=Service_Header, Serv_class=Low)
Server_Add=Rem_Add
Start T1

WAIT-REQ **REQ6** **WAIT-RES**
DDLM_Act_Param.req(Rem_Add,Area_Code,Activate)
=> Service_Header[1]=47H
Service_Header[2]=Area_Code
Service_Header[3]=Activate
FDL_DATA_REPLY.req(SSAP=54, DSAP=54, Rem_add=Rem_Add,
L_sdu=Service_Header, Serv_class=Low)
Server_Add=Rem_Add
Start T1

WAIT-REQ **ACT1** **WAIT-CON**
DDLM_Act_Para_Brct.req(Rem_Add,Area_Code)
=> FDL_DATA.req(SSAP=54, DSAP=54, Rem_add=Rem_Add,
L_sdu[1]=46H, L_sdu[2]=Area_Code, Serv_class=Low)

WAIT-CON **ACT2** **WAIT-REQ**
FDL_DATA.con(SSAP=54)
\L_status=OK/DS
=> DDLM_Act_Para_Brct.con(Status=L_status)

WAIT-CON **FATAL2** **POWER-ON**
FDL_DATA.con(SSAP=54)
\L_status=LS/LR/IV
=> DDLM_Fault.ind

WAIT-RES **POLL1** **WAIT-RES**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=NR
=> FDL_DATA_REPLY.req(SSAP=54, DSAP=54, Rem_add=Server_Add,
L_sdu.len=0, Serv_class=Low)

WAIT-RES **FATAL3** **POWER-ON**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=IV/LS
=> DDLM_Fault.ind

WAIT-RES **FDL1** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DS/NA/UE/RR/RS AND Service_Header[1]=41H
=> DDLM_Get_Master_Diag.con(Status=L_status)
Stop T1

WAIT-RES **FDL2** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DS/NA/UE/RR/RS AND Service_Header[1]=42H
=> DDLM_Start_Seq.con(Status=L_status)
Stop T1

WAIT-RES **FDL3** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DS/NA/UE/RR/RS AND Service_Header[1]=43H
=> DDLM_Download.con(Status=L_status)
Stop T1

WAIT-RES **FDL4** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DS/NA/UE/RR/RS AND Service_Header[1]=44H
=> DDLM_Upload.con(Status=L_status)
Stop T1

WAIT-RES **FDL5** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DS/NA/UE/RR/RS AND Service_Header[1]=45H
=> DDLM_End_Seq.con(Status=L_status)
Stop T1

WAIT-RES **FDL6** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DS/NA/UE/RR/RS AND Service_Header[1]=47H
=> DDLM_Act_Param.con(Status=L_status)
Stop T1

WAIT-RES **RR1** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=RDL/RDH AND Service_Header[1]=41H
=> DDLM_Get_Master_Diag.con(Status=RR)
Stop T1

WAIT-RES **RR2** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=RDL/RDH AND Service_Header[1]=42H
=> DDLM_Start_Seq.con(Status=RR)
Stop T1

WAIT-RES **RR3** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=RDL/RDH AND Service_Header[1]=43H
=> DDLM_Download.con(Status=RR)
Stop T1

WAIT-RES **RR4** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=RDL/RDH AND Service_Header[1]=44H
=> DDLM_Upload.con(Status=RR)
Stop T1

WAIT-RES **RR5** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=RDL/RDH AND Service_Header[1]=45H
=> DDLM_End_Seq.con(Status=RR)
Stop T1

WAIT-RES **RR6** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=RDL/RDH AND Service_Header[1]=47H
=> DDLM_Act_Param.con(Status=RR)
Stop T1

WAIT-RES **TIMO1** **TIMO**
T1 expired
\Service_Header[1]=41H
=> DDLM_Get_Master_Diag.con(Status=TO)
Start T1

WAIT-RES **TIMO2** **TIMO**
T1 expired
\Service_Header[1]=42H
=> DDLM_Start_Seq.con(Status=TO)
Start T1

WAIT-RES **TIMO3** **TIMO**
T1 expired
\Service_Header[1]=43H
=> DDLM_Download.con(Status=TO)
Start T1

WAIT-RES **TIMO4** **TIMO**
T1 expired
\Service_Header[1]=44H
=> DDLM_Upload.con(Status=TO)
Start T1

WAIT-RES **TIMO5** **TIMO**
T1 expired
\Service_Header[1]=45H
=> DDLM_End_Seq.con(Status=TO)
Start T1

WAIT-RES **TIMO6** **TIMO**
T1 expired
\Service_Header[1]=47H
=> DDLM_Act_Param.con(Status=TO)
Start T1

WAIT-RES **REM1** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DL AND L_sdu[1]=FE/NE/AD/EA/LE/RE/IP
AND Service_Header[1]=41H
=> DDLM_Get_Master_Diag.con(Status=L_sdu[1])
Stop T1

WAIT-RES **REM2** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DL AND L_sdu[1]=FE/NE/AD/EA/LE/RE/IP/NI/SE/SC
AND Service_Header[1]=42H
=> DDLM_Start_Seq.con(Status=L_sdu[1])
Stop T1

WAIT-RES **REM3** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DL AND L_sdu[1]=FE/NE/AD/EA/LE/RE/NC/SC/NI
AND Service_Header[1]=43H
=> DDLM_Download.con(Status=L_sdu[1])
Stop T1

WAIT-RES **REM4** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DL AND L_sdu[1]=FE/NE/AD/EA/LE/RE/NI/SC
AND Service_Header[1]=44H
=> DDLM_Upload.con(Status=L_sdu[1])
Stop T1

WAIT-RES **REM5** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DL AND L_sdu[1]=FE/NE/AD/EA/LE/RE/NI/SC
AND Service_Header[1]=45H
=> DDLM_End_Seq.con(Status=L_sdu[1])
Stop T1

WAIT-RES **REM6** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DL AND L_sdu[1]=FE/NE/AD/EA/LE/RE/IP/SC/NI/DI
AND Service_Header[1]=47H
=> DDLM_Act_Param.con(Status=L_sdu[1])
Stop T1

WAIT-RES **RER1** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DH/LR OR L_status=DL
AND L_sdu[1]#FE/NE/AD/EA/LE/RE/IP AND Service_Header[1]=41H
AND (L_sdu.len<3 OR L_sdu[1-2]#Service_Header)
=> DDLM_Get_Master_Diag.con(Status=RE)
Stop T1

WAIT-RES **RER2** **WAIT-REQ**

FDL_DATA_REPLY.con(SSAP=54)
\L_status=DH/LR OR L_status=DL
AND L_sdu[1]#FE/NE/AD/EA/LE/RE/IP/NI/SE/SC
AND Service_Header[1]=42H
AND (L_sdu.len#5 OR L_sdu[1-4]#Service_Header)
=> DDLM_Start_Seq.con(Status=RE)
Stop T1

WAIT-RES **RER3** **WAIT-REQ**

FDL_DATA_REPLY.con(SSAP=54)
\L_status=DH/LR OR L_status=DL
AND L_sdu[1]#FE/NE/AD/EA/LE/RE/NC/SC/NI
AND Service_Header[1]=43H
AND L_sdu#Service_Header
=> DDLM_Download.con(Status=RE)
Stop T1

WAIT-RES **RER4** **WAIT-REQ**

FDL_DATA_REPLY.con(SSAP=54)
\L_status=DH/LR OR L_status=DL
AND L_sdu[1]#FE/NE/AD/EA/LE/RE/NI/SC
AND Service_Header[1]=44H
AND (L_sdu.len<5 OR L_sdu[1-4]#Service_Header[1-4]
OR L_sdu.len>Service_Header[5]+4)
=> DDLM_Upload.con(Status=RE)
Stop T1

WAIT-RES **RER5** **WAIT-REQ**

FDL_DATA_REPLY.con(SSAP=54)
\L_status=DH/LR OR L_status=DL
AND L_sdu[1]#FE/NE/AD/EA/LE/RE/NI/SC
AND Service_Header[1]=45H
AND L_sdu#Service_Header
=> DDLM_End_Seq.con(Status=RE)
Stop T1

WAIT-RES **RER6** **WAIT-REQ**

FDL_DATA_REPLY.con(SSAP=54)
\L_status=DH/LR OR L_status=DL
AND L_sdu[1]#FE/NE/AD/EA/LE/RE/IP/SC/NI/DI
AND Service_Header[1]=47H
AND L_sdu#Service_Header
=> DDLM_Act_Param.con(Status=RE)
Stop T1

WAIT-RES **RES1** **WAIT-REQ**

FDL_DATA_REPLY.con(SSAP=54)
\L_status=DL AND L_sdu[1]#FE/NE/AD/EA/LE/RE/IP
AND Service_Header[1]=41H AND (L_sdu.len>=3
AND L_sdu[1-2]=Service_Header)
=> DDLM_Get_Master_Diag.con(Status=OK,
Diagnostic_Data=L_sdu[3-L_sdu.len])
Stop T1

WAIT-RES **RES2** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DL AND L_sdu[1]#FE/NE/AD/EA/LE/RE/IP/NI/SE/SC
AND Service_Header[1]=42H AND (L_sdu.len=5
AND L_sdu[1-4]=Service_Header)
=> DDLM_Start_Seq.con(Status =OK,
Max_Length_Data_Unit=L_sdu[5])
Stop T1

WAIT-RES **RES3** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DL AND L_sdu[1]#FE/NE/AD/EA/LE/RE/NC/SC/NI
AND Service_Header[1]=43H AND L_sdu=Service_Header
=> DDLM_Download.con(Status=OK)
Stop T1

WAIT-RES **RES4** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DL AND L_sdu[1]#FE/NE/AD/EA/LE/RE/NI/SC
AND Service_Header[1]=44H AND (L_sdu.len>=5
AND L_sdu[1-4]=Service_Header[1-4]
AND L_sdu.len<=Service_Header[5]+4)
=> DDLM_Upload.con(Status=OK,Data=L_sdu[5-L_sdu.len])
Stop T1

WAIT-RES **RES5** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DL AND L_sdu[1]#FE/NE/AD/EA/LE/RE/NI/SE
AND Service_Header[1]=45H AND L_sdu=Service_Header
=> DDLM_End_Seq.con(Status=OK)
Stop T1

WAIT-RES **RES6** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
\L_status=DL AND L_sdu[1]#FE/NE/AD/EA/LE/RE/IP/SC/NI/DI
AND Service_Header[1]=47H AND L_sdu=Service_Header
=> DDLM_Act_Param.con(Status=OK)
Stop T1

TIMO **TOCON1** **WAIT-REQ**
FDL_DATA_REPLY.con(SSAP=54)
=> Stop T1

TIMO **FATAL4** **POWER-ON**
T1 expired
=> DDLM_Fault.ind

13.6 Communication Model of the DP-Slave

13.6.1 General

The DP-Slave is subdivided in four instances:

1. Layer 1/2, Subset of PROFIBUS (FDL and FMA1/2)
2. Direct-Data-Link-Mapper (DDLM)
3. User-Interface
4. User

A summary of the interactions between the individual instances is shown in Figure 39.

13.6.2 FDL and FMA1/2

At minimum, layer 2 shall realize the FDL-services SRD and SDN as responder. In addition to the mandatory services Reset FMA1/2 and Event FMA1/2 the service RSAP Activate FMA1/2 is required as mandatory. Layer 2 shall be able to realize the SAP-access-protection and to perform the Reply-Update-Mode as Multiple.

More hints can be taken from the sections "Transmission Technology" and "Medium Access and Transmission Protocol".

13.6.3 Direct Data Link Mapper

The DDLM includes the mapping of the DDLM-functions onto the layer 2 services.

The DDLM of the DP-Slave shall realize the following local functions as mandatory.

-DDLM_Slave_Init
-DDLM_Fault
-DDLM_Enter
-DDLM_Leave

Additionally the following data transmission services are prescribed:

-DDLM_Slave_Diag
-DDLM_Set_Prm
-DDLM_Chk_Cfg
-DDLM_Data_Exchange
-DDLM_Global_Control
-DDLM_RD_Inp
-DDLM_RD_Outp
-DDLM_Get_Cfg

The following data transmission function can additionally offered from the DDLM:

-DDLM_Set_Slave_Add

The User-Interface uses these functions to handle the data exchange with the DP-Master.

13.6.4 User-Interface

The User-Interface communicates with the DDLM via the functions described in the preceding section.

The interface to the user is realized as a data interface.

The state machine for the slave operations performs the monitoring of the data transfer of this DP-Slave.

The DP-Master shall guarantee that the elapsed time between two passes through the slave-list is not shorter than a projected time interval. This time interval (Min_Slave_Interval) is projected in the master parameter set. This time interval shall be chosen in dependency of the performance of the real DP-Slaves that are included in the DP-System.

The errors of FDL and DDLM which occur locally are analyzed and result in a DDLM_Fault.ind.
The necessary synchronization between the User-Interface and the real application process is realized locally and implementation dependent. It is represented by the local events

- new input data
- change in Diag_Data
- configuration change.

13.6.5 User

The user represents the real application process in the respective device and shall be defined application dependent.

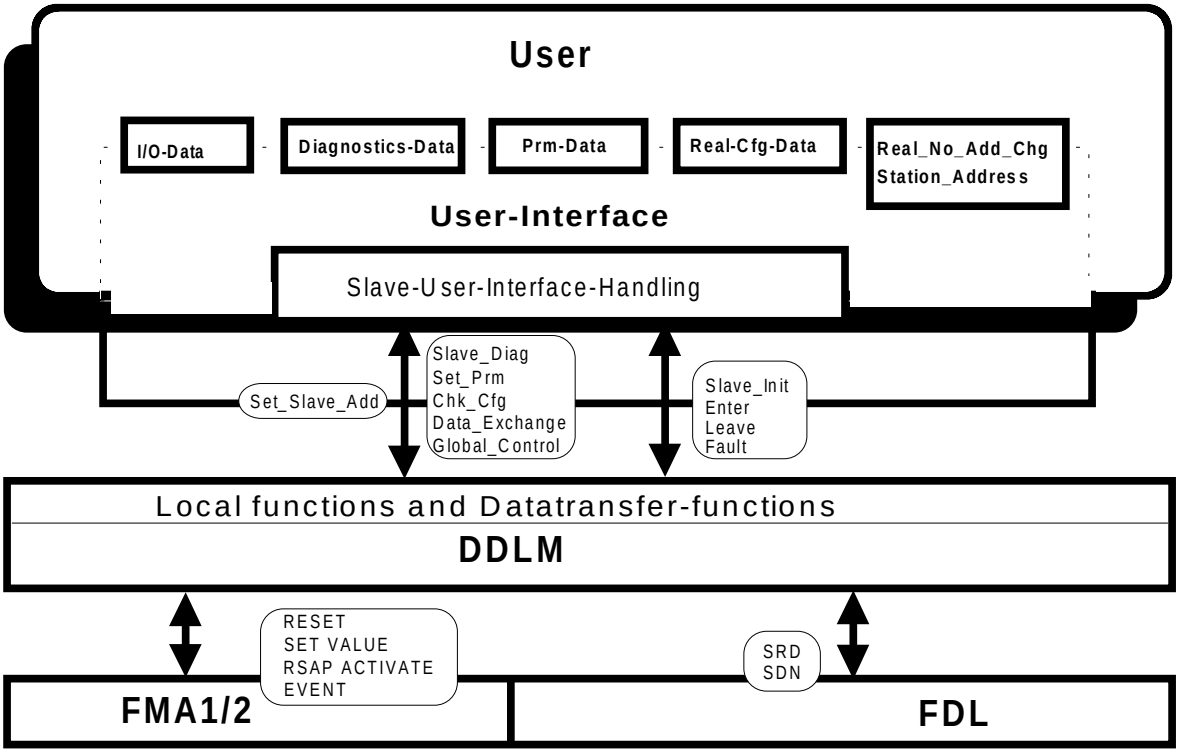


Figure 39. Structure of a DP-Slave

13.7 State Machines in the DP-Slave

13.7.1 State Machine of User-Interface

13.7.1.1 State Diagram of User-Interface

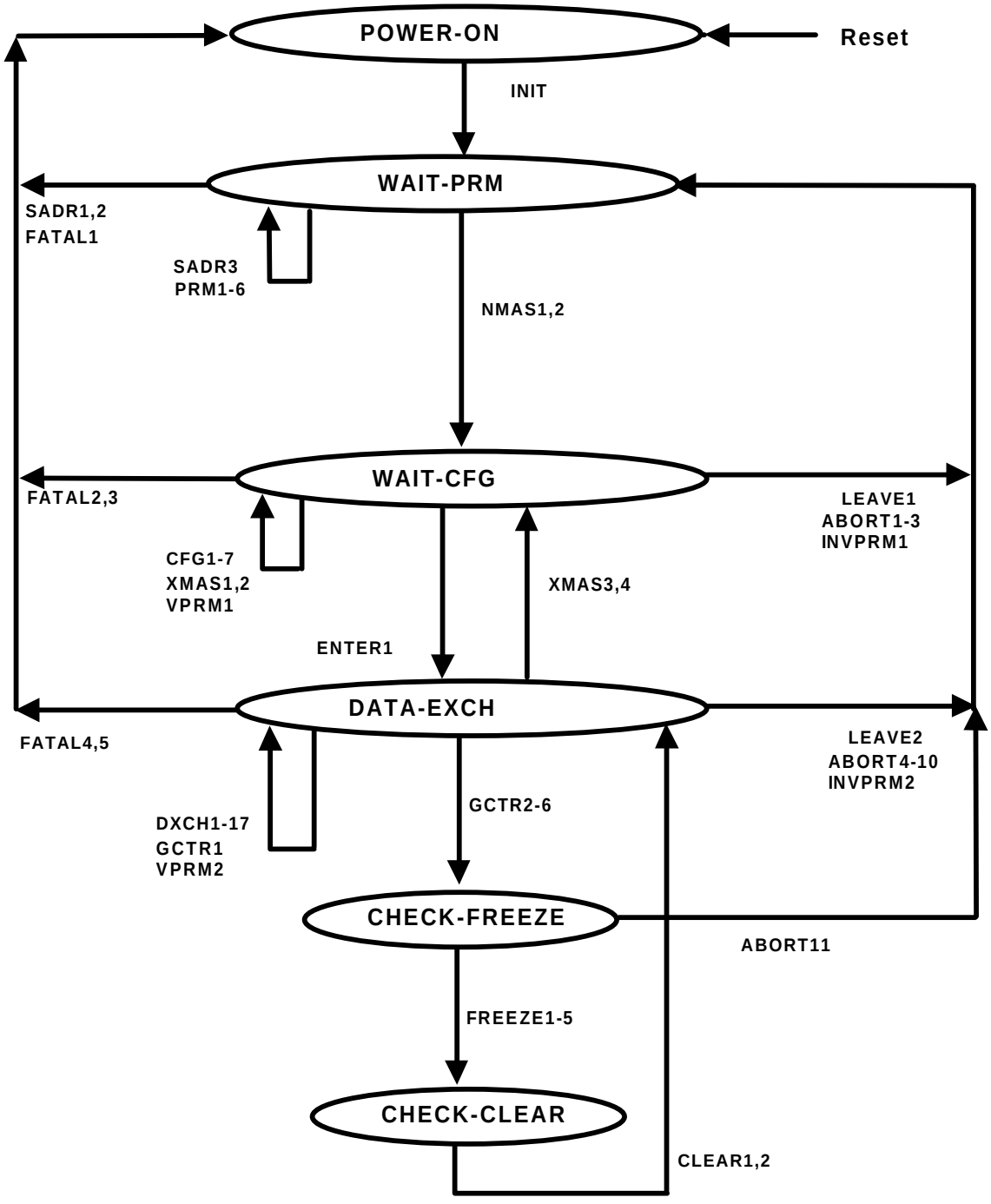


Figure 40. State Diagram for the User-Interface at the DP-Slave

13.7.1.2 State Machine Description

The User-Interface of the DP-Slave has no pronounced interface to the user. The communication takes places via local variables which are described in section a). Furthermore the machine uses functions and macros which are listed in section b) and section c).

The user has the possibility to interact via "local events"

- configuration change
- new input data
- change in Diag_Data

and the capability to initiate a reset.

a) Local variables of the DP-Slave

In order to synchronize the data capturing on the bus, an intermediate storage of the actual data is necessary. To describe the effects of the individual actions in a meaningful way, the buffers are named as follows:

PROFIBUS-DP ->Output-L ->Output-D -> to periphery

PROFIBUS-DP <-Input-D <- periphery-I

Devices which do not support the Sync- respective the Freeze-Mode, are not required to implement the described buffer-model completely.

Output-L:
(Octet-String)
Intermediate storage for outputs.

Output-D:
(Octet-String)
Real output data of the DP-Slave. The requirements for the consistency as described in the Cfg_Data shall be taken into consideration.

Input-D:
(Octet-String)
Intermediate storage for input data. They are stored either by "Freeze" (->DDLM_ Global_Control), or by change of the local input data.

Periphery-I:
(Octet-String)
Depending on the type (see description of Cfg_Data), the data shall be consistently loaded (in Input-D), or may be captured freely.

Station_Address:
(Unsigned8)
Own station address which may be set by ->DDLM_Set_Slave_Address or which is locally stored.

Real_Cfg_Data:
(Octet-String)
The configuration which was previously defined for the device, or the configuration which is determined in the start-up by the device, respectively. The structure of the individual octets corresponds to the structure of Cfg_Data (-> DDLM_Chk_Cfg).

Real_Output_len:
(Unsigned8)
Indicates the number of output bytes that are actually present in the device.

Real_No_Add_Chg:
(Boolean)
Indicates if an address change is possible (False) or not. Can be statically defined or stored.

Real_Ident:
(Unsigned16)
Corresponds to the Ident_Number of parametrization (-> DDLM_Set_Prm).

Active_Groups:
(Unsigned8)
Contains the Group_Ident as set via DDLM_Set_Prm. This is used at DDLM_Global_Control for the decision if the station is addressed.

Diag_Flag:
(Boolean)
This flag indicates that the master shall be informed by an high prior Reply-Update that a change in the diagnostic information occurred.

Sync_Not_Supported:
(Boolean)
Indicates that the DP-Slave does not support the Sync-Mode.

Sync_Supported:
(Boolean)
Indicates that the DP-Slave does support the Sync-Mode.

Freeze_Not_Supported:
(Boolean)
Indicates that the DP-Slave does not support the Freeze-Mode.

Freeze_Supported:
(Boolean)
Indicates that the DP-Slave does support the Freeze-Mode.

b) Functions:

Load WD: (watchdog control)
Loading of the WD-timer with the values (WD_Fact_1, WD_Fact_2).

Start WD: (watchdog control)
Start of the WD-timer (first start).

Stop WD: (watchdog control)
Stop of the WD-timer. After this, the timer is inactive until the next start.

Trigger WD: (watchdog control)
Depending on the state, the WD-timer is started anew (all calls after Start WD), or resumes inactive (all calls after Stop WD). The watchdog is triggered by a valid DDLM_Data_Exchange.ind, DDLM_Slave_Diag.ind and DDLM_Chk_Cfg.ind from the controlling master (see section "Control Intervals").

Valid User_Prm_Data:
User dependent function that tests the User_Prm_Data and returns true if the data is valid.

Invalid User_Prm_Data:
User dependent function that tests the User_Prm_Data and returns true if the data is invalid.

c) Macros:

LEAVE-MASTER

```
      =>    Diag.Master_Add = invalid
            Diag.Sync_Mode = False
            Diag.Freeze_Mode = False
            Diag.Prm_Req = True
            Diag.Station_Not_Ready = True
            Stop WD, Diag.WD_On = False
            Output-D = 0
            Output-L = 0
            DDLM_Leave.Req
            DDLM_Slave_Diag_Upd.req(Diag_Data)
```

13.7.1.3 State Transitions

POWER-ON **INI1** **WAIT-PRM**

```
     \Station_Address setzen
     => DDLM_Slave_Init.req(Station_Address)
        Diag.Prm_Req = True
        Diag.Cfg_Fault = False
        Diag.Prm_Fault = False
        Diag.Not_Supported = False
        Diag.Master_Add = Invalid
        Diag.WD_On = False
        Diag.Station_Not_Ready = True
        Diag.Sync_Mode = False
        Diag.Freeze_Mode = False
        Output_D = 0
        Output_L = 0
        Real_Cfg_Data setzen
        Real_Output_len setzen
        Real_No_Add_Chg setzen
        Real_Ident setzen
        Diag_Flag = False
        DDLM_Slave_Diag_Upd.req(Diag_Data)
        DDLM_Get_Cfg_Upd.req(Real_Cfg_Data)
```

WAIT-PRM **SADR1** **POWER-ON**

DDLM_Set_Slave_Add.ind

```
     \Real_No_Add_Chg = False
      AND Real_Ident = Ident_Number
      AND New_Slave_Address <= 125
     => Station_Address = New_Slave_Add
        Real_No_Add_Chg = No_Add_Chg
        Store Station_Address, No_Add_Chg and Rem_Slave_Data
```

WAIT-PRM **SADR2** **WAIT-PRM**

DDLM_Set_Slave_Add.ind

```
\Real_No_Add_Chg = True
 OR Real_Ident # Ident_Number
 OR New_Slave_Add > 125
=> ignore
```

WAIT-PRM **PRM1** **WAIT-PRM**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Lock_Req = False AND Unlock_Req = False
=> DDLM_Set_minTsdr.req(min_Tsdr)
```

WAIT-PRM **PRM2** **WAIT-PRM**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Unlock_Req = True
=> ignore
```

WAIT-PRM **PRM3** **WAIT-PRM**

DDLM_Chk_Cfg.ind

```
=> ignore
```

WAIT-PRM **PRM4** **WAIT-PRM**

DDLM_Slave_Diag.ind

```
=> ignore
```

WAIT-PRM **PRM5** **WAIT-PRM**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Lock_Req = True
 AND Unlock_Req = False AND (Ident_Number # Real_Ident
 OR WD_On = True AND (WD_Fact_1=0 OR WD_Fact_2=0))
=> Diag.Prm_Fault = True
   DDLM_Slave_Diag_Upd.req(Diag_Data)
```

WAIT-PRM **PRM6** **WAIT-PRM**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Lock_Req = True AND Unlock_Req = False
 AND Ident_Number = Real_Ident
 AND (WD_On = False OR (WD_Fact_1 > 0 AND WD_Fact_2 > 0))
 AND (Freeze_Req = True AND Freeze_Not_Supported
 OR Sync_Req = True AND Sync_Not_Supported
 OR Prm_Data[1].0, .1, .2 = True)
=> Diag.Not_Supported = True
   DDLM_Slave_Diag_Upd.req(Diag_Data)
```

WAIT-PRM **FATAL1** **POWER-ON**

DDLM_Fault.ind

WAIT-PRM **NMAS1** **WAIT-CFG**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Lock_Req = True AND Unlock_Req = False
 AND Ident_Number = Real_Ident
 AND WD_On = True AND WD_Fact_1 > 0 AND WD_Fact_2 > 0
 AND (Freeze_Req = False OR Freeze_Supported)
 AND (Sync_Req = False OR Sync_Supported)
 AND (Prm_Data[1].0, .1, .2 = False)
=> Diag.Master_Add = Req_Add
    Load WD, Start WD, Diag.WD_On = True
    DDLM_Set_minTsdr.req(minTsdr)
    Active_Groups = Group_Ident
    Diag.Prm_Fault = False
    Diag.Prm_Req = False
    Diag.Not_Supported = False
    DDLM_Slave_Diag_Upd.req(Diag_Data)
    check User_Prm_Data
```

WAIT-PRM **NMAS2** **WAIT-CFG**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Lock_Req = True AND Unlock_Req = False
 AND (Ident_Number = Real_Ident)
 AND WD_On = False
 AND (Freeze_Req = False OR Freeze_Supported)
 AND (Sync_Req = False OR Sync_Supported)
 AND (Prm_Data[1].0, .1, .2 = False)
=> Diag.Master_Add = Req_Add
    DDLM_Set_minTsdr.req(minTsdr)
    Active_Groups = Group_Ident
    Diag.Prm_Fault = False
    Diag.Prm_Req = False
    Diag.Not_Supported = False
    DDLM_Slave_Diag_Upd.req(Diag_Data)
    check User_Prm_Data
```

WAIT-CFG **INVPRM1** **WAIT-PRM**

```
\invalid User_Prm_Data
=> Diag.Prm_Fault = True
    Diag.Master_Add = Invalid
    Diag.Prm_Req = True
    Stop WD, Diag.WD_On = False
    DDLM_Slave_Diag_Upd.req(Diag_Data)
```

WAIT-CFG **VPRM1** **WAIT-CFG**

```
\valid User_Prm_Data
=> ignore
```

WAIT-CFG **LEAVE1** **WAIT-PRM**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Unlock_Req = True
 AND Diag.Master_Add = Req_Add
=> Diag.Master_Add = Invalid
    Diag.Prm_Req = True
    Stop WD, Diag.WD_On = False
    DDLM_Slave_Diag_Upd.req(Diag_Data)
```

WAIT-CFG **CFG1** **WAIT-CFG**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Unlock_Req = True
 AND Diag.Master_Add # Req_Add
=>  ignore
```

WAIT-CFG **ABORT1** **WAIT-PRM**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Unlock_Req = False AND Lock_Req = True
 AND Diag.Master_Add = Req_Add
 AND (Ident_Number # Real_Ident
 OR WD_On = True AND (WD_Fact_1=0 OR WD_Fact_2=0))
=>  Diag.Prm_Fault = True
    Diag.Master_Add = Invalid
    Diag.Prm_Req = True
    Stop WD, Diag.WD_On = False
    DDLM_Slave_Diag_Upd.req(Diag_Data)
```

WAIT-CFG **ABORT2** **WAIT-PRM**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Unlock_Req = False AND Lock_Req = True
 AND Diag.Master_Add = Req_Add
 AND Ident_Number = Real_Ident
 AND (WD_On = False OR WD_Fact_1>0 AND WD_Fact_2>0)
 AND (Freeze_Req = True AND Freeze_Not_Supported
 OR Sync_Req = True AND Sync_Not_Supported
 OR Prm_Data[1].0, .1, .2 = True)
=>  Diag.Not_Supported = True
    Diag.Master_Add = Invalid
    Diag.Prm_Req = True
    Stop WD, Diag.WD_On = False
    DDLM_Slave_Diag_Upd.req(Diag_Data)
```

WAIT-CFG **CFG2** **WAIT-CFG**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Unlock_Req = False AND Lock_Req = True
 AND Diag.Master_Add # Req_Add
 AND (Ident_Number # Real_Ident
 OR WD_On = True AND (WD_Fact_1=0 OR WD_Fact_2=0)
 OR Freeze_Req = True AND Freeze_Not_Supported
 OR Sync_Req = True AND Sync_Not_Supported
 OR Prm_Data[1].0, .1, .2 = True)
=>  ignore
```

WAIT-CFG **XMAS1** **WAIT-CFG**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Unlock_Req = False AND Lock_Req = True
 AND Ident_Number = Real_Ident
 AND WD_On = True AND WD_Fact_1 > 0 AND WD_Fact_2 > 0
 AND (Freeze_Req = False OR Freeze_Supported)
 AND (Sync_Req = False OR Sync_Supported)
 AND (Prm_Data[1].0, .1, .2 = False)
=>  Diag.Master_Add = Req_Add
    Load WD, Start WD, Diag.WD_On = True
    Active_Groups = Group_Ident
    DDLM_Set_minTsdr.req(minTsdr)
    DDLM_Slave_Diag_Upd.req(Diag_Data)
    check User_Prm_Data
```

WAIT-CFG **XMAS2** **WAIT-CFG**
DDLM_Set_Prm.ind
\Prm_Data.len ≥ 7 AND Unlock_Req = False AND Lock_Req = True
AND Ident_Number = Real_Ident
AND WD_On = False
AND (Freeze_Req = False OR Freeze_Supported)
AND (Sync_Req = False OR Sync_Supported)
AND (Prm_Data[1].0, .1, .2 = False)
=> Diag.Master_Add = Req_Add
Stop WD, Diag.WD_On = False
Active_Groups = Group_Ident
DDLM_Set_minTsdr.req(minTsdr)
DDLM_Slave_Diag_Upd.req(Diag_Data)
check User_Prm_Data

WAIT-CFG **CFG3** **WAIT-CFG**
DDLM_Set_Prm.ind
\Prm_Data.len ≥ 7 AND Unlock_Req = False AND Lock_Req = False
=> DDLM_Set_minTsdr.req(minTsdr)

WAIT-CFG **CFG4** **WAIT-CFG**
DDLM_Slave_Diag.ind
\Diag.Master_Add = Req_Add
=> WD triggern

WAIT-CFG **CFG5** **WAIT-CFG**
DDLM_Slave_Diag.ind
\Diag.Master_Add # Req_Add
=> ignore

WAIT-CFG **CFG6** **WAIT-CFG**
DDLM_Chk_Cfg.ind
\Diag.Master_Add # Req_Add
=> ignore

WAIT-CFG **CFG7** **WAIT-CFG**
DDLM_Set_Slave_Add.ind
=> ignore

WAIT-CFG **ABORT3** **WAIT-PRM**
DDLM_Chk_Cfg.ind
\Diag.Master_Add = Req_Add AND Cfg_Data # Real_Cfg_Data
=> Diag.Cfg_Fault = True
Diag.Prm_Req = True
Diag.Master_Add = Invalid
Stop WD, Diag.WD_On = False
DDLM_Slave_Diag_Upd.req(Diag_Data)

WAIT-CFG **FATAL2** **POWER-ON**
WD_Time_Out

WAIT-CFG **FATAL3** **POWER-ON**
DDLM_Fault.ind

WAIT-CFG **ENTER1** **DATA-EXCH**

DDLM_Chk_Cfg.ind

```
\Diag.Master_Add = Req_Add AND Cfg_Data = Real_Cfg_Data
=>  DDLM_Enter(Diag.Master_Add)
    Diag.Cfg_Fault = False
    Diag_Flag = True
    Diag.Station_Not_Ready = False
    DDLM_RD_Outp_Upd.req(Output-D)
    Input-D = Peripherie-I
    DDLM_RD_Inp_Upd.req(Input-D)
    DDLM_Data_Exchange_Upd.req(Diag_Flag,Input-D)
    DDLM_Slave_Diag_Upd.req(Diag_Data)
    WD triggern
```

DATA-EXCH **DXCH1** **DATA-EXCH**

DDLM_Data_Exchange.ind

```
\Outp_Data.len > 0 AND Outp_Data.len=real_Output_len
 AND Diag.Sync_Mode = False
=>  Output-L = Output-D = Outp-Data
    DDLM_RD_Outp_Upd.req(Output-D)
    WD triggern
```

DATA-EXCH **DXCH2** **DATA-EXCH**

DDLM_Data_Exchange.ind

```
\Outp_Data.len > 0 AND Outp_Data.len=real_Output_len
 AND Diag.Sync_Mode = True
=>  Output-L = Outp-Data
    WD triggern
```

DATA-EXCH **DXCH3** **DATA-EXCH**

DDLM_Data_Exchange.ind

```
\Outp_Data.len = 0 AND real_Output_len = 0
=>  WD triggern
```

DATA-EXCH **ABORT4** **WAIT-PRM**

DDLM_Data_Exchange.ind

```
\Outp_Data.len # real_Output_len
=>  Diag.Cfg_Fault = True
    LEAVE-MASTER
```

DATA-EXCH **DXCH4** **DATA-EXCH**

lokales Ereignis "neue Eingangsdaten"

```
\Diag.Freeze_Mode = False
=>  Input-D = Peripherie-I
    DDLM_Data_Exchange_Upd.req(Diag_Flag, Input-D)
    DDLM_RD_Inp_Upd.req(Input-D)
```

DATA-EXCH **DXCH5** **DATA-EXCH**

lokales Ereignis "neue Eingangsdaten"

```
\Diag.Freeze_Mode = True
=>  ignore
```

DATA-EXCH **DXCH6** **DATA-EXCH**

lokales Ereignis "Änderung in Diag_Data"

```
\Diag.Prm_Req = False
=>  DDLM_Slave_Diag_Upd.req(Diag_Data)
    Diag_Flag=True
    DDLM_Data_Exchange_Upd.req(Diag_Flag, Input-D)
```

DATA-EXCH **ABORT5** **WAIT-PRM**
lokales Ereignis "Änderung in Diag_Data"
\Diag.Prm_Req = True
=> LEAVE-MASTER

DATA-EXCH **DXCH7** **DATA-EXCH**
DDLM_Slave_Diag.ind
\Diag.Master_Add = Req_Add AND Diag.Stat_Diag = True
=> WD triggern

DATA-EXCH **DXCH8** **DATA-EXCH**
DDLM_Slave_Diag.ind
\Diag.Master_Add = Req_Add AND Diag.Stat_Diag = False
=> Diag_Flag=False
DDLM_Data_Exchange_Upd.req(Diag_Flag, Input-D)
WD triggern

DATA-EXCH **DXCH9** **DATA-EXCH**
DDLM_Slave_Diag.ind
\Diag.Master_Add # Req_Add
=> ignore

DATA-EXCH **DXCH10** **DATA-EXCH**
DDLM_Chk_Cfg.ind
\Diag.Master_Add # Req_Add
=> ignore

DATA-EXCH **DXCH11** **DATA-EXCH**
DDLM_Chk_Cfg.ind
\Diag.Master_Add = Req_Add AND Cfg_Data = Real_Cfg_Data
=> WD triggern

DATA-EXCH **ABORT6** **WAIT-PRM**
DDLM_Chk_Cfg.ind
\Diag.Master_Add = Req_Add AND Cfg_Data # Real_Cfg_Data
=> Diag.Cfg_Fault = True
LEAVE-MASTER

DATA-EXCH **ABORT7** **WAIT-PRM**
lokales Ereignis "Konfigurationsänderung"
=> Diag.Cfg_Fault = True
DDLM_Get_Cfg_Upd.req(Real_Cfg_Data)
LEAVE-MASTER

DATA-EXCH **DXCH12** **DATA-EXCH**
DDLM_Set_Prm.ind
\Prm_Data.len ≥ 7 AND Lock_Req = False AND Unlock_Req = False
=> DDLM_Set_minTsdr.req(minTsdr)

DATA-EXCH **DXCH13** **DATA-EXCH**
DDLM_Set_Prm.ind
\Prm_Data.len ≥ 7 AND Unlock_Req = True
AND Diag.Master_Add # Req_Add
=> ignore

DATA-EXCH **LEAVE2** **WAIT-PRM**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Unlock_Req = True
 AND Diag.Master_Add = Req_Add
=>  LEAVE-MASTER
```

DATA-EXCH **ABORT8** **WAIT-PRM**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Lock_Req = True AND Unlock_Req = False
 AND Diag.Master_Add = Req_Add
 AND (Ident_Number # Real_Ident
 OR WD_On = True AND (WD_Fact_1=0 OR WD_Fact_2=0))
=>  Diag.Prm_Fault = True
    LEAVE-MASTER
```

DATA-EXCH **ABORT9** **WAIT-PRM**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Lock_Req = True AND Unlock_Req = False
 AND Diag.Master_Add = Req_Add
 AND Ident_Number = Real_Ident
 AND (WD_On = False OR WD_Fact_1>0 AND WD_Fact_2>0)
 AND (Freeze_Req = True AND Freeze_Not_Supported
 OR Sync_Req = True AND Sync_Not_Supported
 OR Prm_Data[1].0, .1, .2 = True)
=>  Diag.Not_Supported = True
    LEAVE-MASTER
```

DATA-EXCH **DXCH14** **DATA-EXCH**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Lock_Req = True AND Unlock_Req = False
 AND Diag.Master_Add # Req_Add
 AND (Ident_Number # Real_Ident
 OR WD_On = True AND (WD_Fact_1=0 OR WD_Fact_2=0)
 OR Freeze_Req = True AND Freeze_Not_Supported
 OR Sync_Req = True AND Sync_Not_Supported
 OR Prm_Data[1].0, .1, .2 = True)
=>  ignore
```

DATA-EXCH **DXCH15** **DATA-EXCH**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Lock_Req = True AND Unlock_Req = False
 AND Diag.Master_Add = Req_Add
 AND Ident_Number = Real_Ident
 AND WD_On = True AND WD_Fact_1>0 AND WD_Fact_2>0
 AND (Freeze_Req = False OR Freeze_Supported)
 AND (Sync_Req = False OR Sync_Supported)
 AND (Prm_Data[1].0, .1, .2 = False)
=>  Load WD, Start WD, Diag.WD_On = True
    Active_Groups = Group_Ident
    DDLM_Slave_Diag_Upd.req(Diag_Data)
    DDLM_Set_minTsdr(minTsdr)
    check User_Prm_Data
```

DATA-EXCH **DXCH16** **DATA-EXCH**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Lock_Req = True AND Unlock_Req = False
 AND Diag.Master_Add = Req_Add
 AND Ident_Number = Real_Ident
 AND WD_On = False
 AND (Freeze_Req = False OR Freeze_Supported)
 AND (Sync_Req = False OR Sync_Supported)
 AND (Prm_Data[1].0, .1, .2 = False)
=> Stop WD, Diag.WD_On = False
   Active_Groups = Group_Ident
   DDLM_Slave_Diag_Upd.req(Diag_Data)
   DDLM_Set_minTsdr(minTsdr)
   check User_Prm_Data
```

DATA-EXCH **INVPRM2** **WAIT-PRM**

```
\invalid User_Prm_Data
=> Diag.Prm_Fault = True
   LEAVE-MASTER
```

DATA-EXCH **VPRM2** **DATA-EXCH**

```
\valid User_Prm_Data
=> ignore
```

DATA-EXCH **XMAS3** **WAIT-CFG**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Lock_Req = True AND Unlock_Req = False
 AND Diag.Master_Add # Req_Add AND Ident_Number = Real_Ident
 AND WD_On = True AND WD_Fact_1 > 0 AND WD_Fact_2 > 0
 AND (Freeze_Req = False OR Freeze_Supported)
 AND (Sync_Req = False OR Sync_Supported)
 AND (Prm_Data[1].0, .1, .2 = False)
=> Diag.Sync_Mode = False
   Diag.Freeze_Mode = False
   Diag.Station_Not_Ready = True
   Load WD, Start WD, Diag.WD_On = True
   Active_Groups = Group_Ident
   DDLM_Leave.req
   Output-D = 0
   Output-L = 0
   DDLM_Set_minTsdr(minTsdr)
   Diag.Master_Add = Req_Add
   DDLM_Slave_Diag_Upd.req(Diag_Data)
   check User_Prm_Data
```

DATA-EXCH **XMAS4** **WAIT-CFG**

DDLM_Set_Prm.ind

```
\Prm_Data.len ≥ 7 AND Lock_Req = True AND Unlock_Req = False
 AND Diag.Master_Add # Req_Add AND Ident_Number = Real_Ident
 AND AND WD_On = False
 AND (Freeze_Req = False OR Freeze_Supported)
 AND (Sync_Req = False OR Sync_Supported)
 AND (Prm_Data[1].0, .1, .2 = False)
=>  Diag.Sync_Mode = False
    Diag.Freeze_Mode = False
    Diag.Station_Not_Ready = True
    Stop WD, Diag.WD_On = False
    Active_Groups = Group_Ident
    DDLM_Leave.req
    Output-D = 0
    Output-L = 0
    DDLM_Set_minTsdr(minTsdr)
    Diag.Master_Add = Req_Add
    DDLM_Slave_Diag_Upd.req(Diag_Data)
    check User_Prm_Data
```

DATA-EXCH **GCTR1** **DATA-EXCH**

DDLM_Global_Control.ind

```
\Req_Add # Diag.Master_Add
 OR Group_Select # 0 AND ((Active_Groups & Group_Select) = 0)
=>  ignore
```

DATA-EXCH **GCTR2** **CHECK-FREEZE**

DDLM_Global_Control.ind

```
\Req_Add = Diag.Master_Add
 AND (Group_Select = 0 OR (Active_Groups & Group_Select) # 0)
 AND Unsync = True AND Sync_Not_Supported
 AND Control_Command.0, .6, .7 = False
```

DATA-EXCH **ABORT10** **WAIT-PRM**

DDLM_Global_Control.ind

```
\Req_Add = Diag.Master_Add
 AND (Group_Select = 0 OR (Active_Groups & Group_Select) # 0)
 AND (Sync = True AND Sync_Not_Supported
 OR Control_Command.0, .6, .7 = True)
=>  Diag.Not_Supported = True
    Leave-Master
```

DATA-EXCH **GCTR3** **CHECK-FREEZE**

DDLM_Global_Control.ind

```
\Req_Add = Diag.Master_Add
 AND (Group_Select = 0 OR (Active_Groups & Group_Select) # 0)
 AND Unsync = True AND Sync_Supported
 AND Control_Command.0, .6, .7 = False
=>  Diag.Sync_Mode = False
    Output-D = Output-L
    DDLM_RD_Outp_Upd.req(Output-D)
    DDLM_Slave_Diag_Upd.req(Diag_Data)
```

DATA-EXCH **GCTR4** **CHECK-FREEZE**
DDLM_Global_Control.ind
\Req_Add = Diag.Master_Add
AND (Group_Select = 0 OR (Active_Groups & Group_Select) # 0)
AND Unsync = False AND Sync = True AND Sync_Supported
AND Diag.Sync_Mode = False
AND Control_Command.0, .6, .7 = False
=> Diag.Sync_Mode = True
DDLM_Slave_Diag_Upd.req(Diag_Data)

DATA-EXCH **GCTR5** **CHECK-FREEZE**
DDLM_Global_Control.ind
\Req_Add = Diag.Master_Add
AND (Group_Select = 0 OR (Active_Groups & Group_Select) # 0)
AND Unsync = False AND Sync = True AND Sync_Supported
AND Diag.Sync_Mode = True
AND Control_Command.0, .6, .7 = False
=> Output-D = Output-L
DDLM_RD_Outp_Upd.req(Output-D)

DATA-EXCH **GCTR6** **CHECK-FREEZE**
DDLM_Global_Control.ind
\Req_Add = Diag.Master_Add
AND (Group_Select = 0 OR (Active_Groups & Group_Select) # 0)
AND Unsync = False AND Sync = False
AND Control_Command.0, .6, .7 = False

CHECK-FREEZE **FREEZE1** **CHECK-CLEAR**
\Unfreeze = True AND Freeze_Not_Supported

CHECK-FREEZE **ABORT11** **WAIT-PRM**
\Unfreeze = False AND Freeze = True AND Freeze_Not_Supported
=> Diag.Not_Supported = True
LEAVE-MASTER

CHECK-FREEZE **FREEZE2** **CHECK-CLEAR**
\Unfreeze = True AND Freeze_Supported
=> Diag.Freeze_Mode = False
DDLM_Slave_Diag_Upd.req(Diag_Data)
Input-D = Peripherie-I
DDLM_Data_Exchange_Upd.req(Diag_Flag, Input-D)
DDLM_RD_Inp_Upd.req(Input-D)

CHECK-FREEZE **FREEZE3** **CHECK-CLEAR**
\Unfreeze = False AND Freeze = True AND Freeze_Supported
AND Diag.Freeze_Mode = False
=> Diag.Freeze_Mode = True
DDLM_Slave_Diag_Upd.req(Diag_Data)
Input-D = Peripherie-I
DDLM_Data_Exchange_Upd.req(Diag_Flag, Input-D)
DDLM_RD_Inp_Upd.req(Input-D)

CHECK-FREEZE **FREEZE4** **CHECK-CLEAR**
\Unfreeze = False AND Freeze = True AND Freeze_Supported
AND Diag.Freeze_Mode = True
=> Input-D = Peripherie-I
DDLM_Data_Exchange_Upd.req(Diag_Flag, Input-D)
DDLM_RD_Inp_Upd.req(Input-D)

CHECK-FREEZE **FREEZE5** **CHECK-CLEAR**
\Unfreeze = False AND AND Freeze = False

CHECK-CLEAR **CLEAR1** **DATA-EXCH**
\Clear_Data = False

CHECK-CLEAR **CLEAR2** **DATA-EXCH**
\Clear_Data = True
=> Output-L = 0
Output-D = 0
DDLM_RD_Outp_Upd.req(Output-D)

DATA-EXCH **DXCH17** **DATA-EXCH**
DDLM_Set_Slave_Add.ind
=> ignore

DATA-EXCH **FATAL4** **POWER-ON**
WD_Time_Out

DATA-EXCH **FATAL5** **POWER-ON**
DDLM_Fault.ind

any-state **LERR1** **any-state**
DDLM_Set_Prm.ind
\Prm_Data.len < 7
=> ignore

13.7.2 State Machine of DDLM

13.7.2.1 State Diagram of DDLM

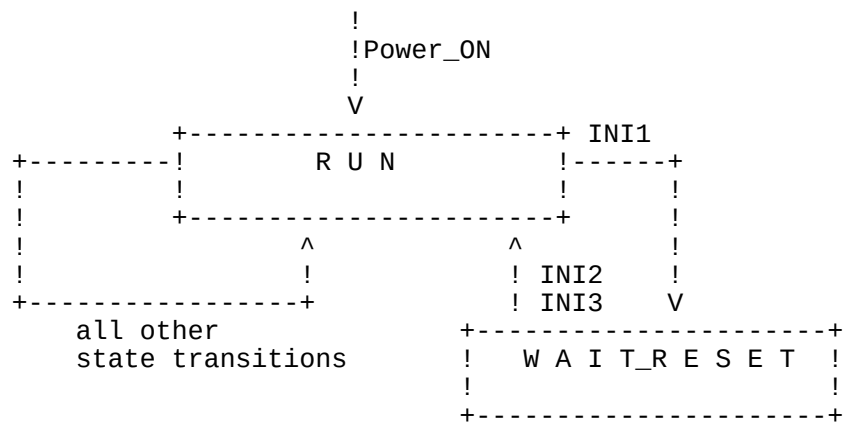


Figure 41. State Diagram of the DDLM at the DP-Slave

13.7.2.2 State Machine Description

The following state machine defines the behaviour of the DDLM at the DP-Slave.

It shall be noticed that the DP-Slave handles high- and low priority indications in the same way.

Basically, the DDLM of the DP-Slave is in the "RUN" state. During initialization of the DDLM and the FDL, the state changes to the "WAIT-RESET" state.

After power-on, the state machine changes to the "RUN" state. In the start-up phase of the state machine the layer 2 bus parameters will be delivered by the DDLM. In implementations where the layer 2 bus parameters are locally known, the respective actions in the state transitions INI1 and INI3 may be omitted.

This state machine assumes that FDL and FMA1/2 are able to accept the necessary number of parallel service requests. With the exception of FDL indications, these are local interactions between DDLM and FDL and FMA1/2, respectively.

During the SAP activation in the state transitions INI3 and INI4, the maximum possible L_sdu-lengths of the various slaves are adjusted. The L_sdu-lengths result from the device configuration. It is acceptable that the L_sdu-length is configured longer than really needed.

The following variables are used for the description of the state machines:

Loc_Station_Address
(Unsigned8)
Local variable for the intermediate storage of the own station address which was delivered with the last DDLM_Slave_Init.

Loc_Baudrate
(Unsigned8)
Local variable for the intermediate storage of the baud rate.

Loc_Medium_red
(Unsigned8)
Local variable for the intermediate storage of parameter redundancy.

Prev_Diag_Flag
(Boolean)
Indicates the state of the Diag_Flag at the last DDLM_Data_Exchange_Update.req. By means of this flag it is detected whether a high prior update is present.

13.7.2.3 State Transitions

RUN **INI1** **WAIT-RESET**
DDLM_Slave_Init.req(Station_Address)
=> FMA1/2_RESET.req
Loc_Station_Address = Station_Address

WAIT-RESET **INI2** **RUN**
FMA1/2_RESET.con(NO/IV)
=> DDLM_Fault.ind

WAIT-RESET **INI3** **RUN**
FMA1/2_RESET.con(OK)
=> FMA1/2_SET_VALUE.req(Variable_name1=TS,
Desired_value1=Loc_Station_Address)
FMA1/2_RSAP_ACTIVATE.req(SSAP=60, Access=All,
Indication_Mode=All)
FMA1/2_RSAP_ACTIVATE.req(SSAP=61, Access=All,
Indication_Mode=All)
FMA1/2_RSAP_ACTIVATE.req(SSAP=62, Access=All,
Indication_Mode=All)
FMA1/2_RSAP_ACTIVATE.req(SSAP=59, Access=All,
Indication_Mode=Data)
FMA1/2_RSAP_ACTIVATE.req(SSAP=55, Access=All,
Indication_Mode=Data)

RUN **INI4** **RUN**
DDLM_Enter.req(Master_Add)
=> FMA1/2_RSAP_ACTIVATE.req(SSAP=NIL, Access = Master_Add,
Indication_Mode=All)
FMA1/2_SAP_ACTIVATE.req(SSAP=58, Access = Master_Add,
Service=SDN, Role_in_service=Responder)
FMA1/2_RSAP_ACTIVATE.req(SSAP=56, Access=All,
Indication_Mode=Data)
FMA1/2_RSAP_ACTIVATE.req(SSAP=57, Access=All,
Indication_Mode=Data)
Prev_Diag_Flag = False

RUN **INI5** **RUN**
DDLM_Leave.req
=> FMA1/2_SAP_DEACTIVATE.req(SSAP=NIL)
FMA1/2_SAP_DEACTIVATE.req(SSAP=56)
FMA1/2_SAP_DEACTIVATE.req(SSAP=58)
FMA1/2_SAP_DEACTIVATE.req(SSAP=57)

RUN **INI6** **RUN**
DDLM_Set_minTsdr.req(minTsdr)
\MinTsdr = 0
=> ignore

RUN **INI7** **RUN**
DDLM_Set_minTsdr.req(minTsdr)
\MinTsdr > 0
=> FMA1/2_SET_VALUE.req(Variable_name1=$minT_{SDR}$,
Desired_value1=minTsdr)

RUN **INI8** **RUN**
FMA1/2_SET_VALUE.con(OK)
=> ignore

RUN **INI9** **RUN**
FMA1/2_SET_VALUE.con(IV/NO)
=> DDLM_Fault.ind

RUN **INI10** **RUN**
FMA1/2_RSAP_ACTIVATE.con(OK)
=> ignore

RUN **INI11** **RUN**
FMA1/2_SAP_ACTIVATE.con(OK)
=> ignore

RUN **INI12** **RUN**
FMA1/2_SAP_DEACTIVATE.con(OK)
=> ignore

RUN **INI13** **RUN**
FMA1/2_RSAP_ACTIVATE.con(IV/NO)
=> DDLM_Fault.ind

RUN **INI14** **RUN**
FMA1/2_SAP_ACTIVATE.con(IV/NO)
=> DDLM_Fault.ind

RUN **INI15** **RUN**
FMA1/2_SAP_DEACTIVATE.con(IV/NO)
=> DDLM_Fault.ind

RUN **DIAG1** **RUN**
DDLM_Slave_Diag_Upd.req(Diag_Data)
=> FDL_REPLY_UPDATE.req(SSAP=60, L_sdu=Diag_Data,
Serv_Class=Low, Transmit=Multiple)

RUN **DIAG2** **RUN**
FDL_DATA_REPLY.ind(DSAP=60)
=> DDLM_Slave_Diag.ind(Req_Add=Loc_add)

RUN **DATA1** **RUN**
DDLM_Data_Exchange_Upd.req(Diag_Flag, Inp_Data)
\Diag_Flag = False AND Prev_Diag_Flag = False
AND Inp_Data.len > 0
=> FDL_REPLY_UPDATE.req(SSAP=NIL, L_sdu=Inp_Data,
Serv_Class=Low, Transmit=Multiple)

RUN **DATA2** **RUN**
DDLM_Data_Exchange_Upd.req(Diag_Flag, Inp_Data)
\Diag_Flag = False AND Prev_Diag_Flag = False
AND Inp_Data.len = 0
=> ignore

RUN **DATA3** **RUN**
DDLM_Data_Exchange_Upd.req(Diag_Flag, Inp_Data)
\Diag_Flag = False AND Prev_Diag_Flag = True
AND Inp_Data.len > 0
=> Prev_Diag_Flag = False
FDL_REPLY_UPDATE.req(SSAP=NIL, L_sdu=Inp_Data,
Serv_Class=Low, Transmit=Multiple)
FDL_REPLY_UPDATE.req(SSAP=NIL, L_sdu.len=0,
Serv_Class=High, Transmit=Single)

RUN **DATA4** **RUN**
DDLM_Data_Exchange_Upd.req(Diag_Flag, Inp_Data)
\Diag_Flag = False AND Prev_Diag_Flag = True
AND Inp_Data.len = 0
=> FDL_REPLY_UPDATE.req(SSAP=NIL, L_sdu.len=0,
Serv_Class=High, Transmit=Single)
Prev_Diag_Flag = False

```
RUN                                  DATA5                                  RUN
    DDLM_Data_Exchange_Upd.req(Diag_Flag, Inp_Data)
     \Diag_Flag = True AND Inp_Data.len > 0
     =>  FDL_REPLY_UPDATE.req(SSAP=NIL, L_sdu=Inp_Data,
           Serv_Class=High, Transmit=Multiple)
         Prev_Diag_Flag = True

RUN                                  DATA6                                  RUN
    DDLM_Data_Exchange_Upd.req(Diag_Flag, Inp_Data)
     \Diag_Flag = True AND Inp_Data.len = 0
     =>  FDL_REPLY_UPDATE.req(SSAP=NIL, L_sdu.len=1, L_sdu[1] = 0,
           Serv_Class=High, Transmit=Multiple)
         Prev_Diag_Flag = True

RUN                                  DATA7                                  RUN
    FDL_DATA_REPLY.ind(DSAP=NIL)
     =>  DDLM_Data_Exchange.ind(Outp_Data=L_sdu)

RUN                                  INP1                                   RUN
    DDLM_RD_Inp_Upd.req(Inp_Data)
     \Inp_Data.len > 0
     =>  FDL_REPLY_UPDATE.req(SSAP=56, L_sdu=Inp_Data,
           Serv_Class=Low, Transmit=Multiple)

RUN                                  INP2                                   RUN
    DDLM_RD_Inp_Upd.req(Inp_Data)
     \Inp_Data.len = 0
     =>  ignore

RUN                                  INP3                                   RUN
    FDL_DATA_REPLY.ind(DSAP=56)
     =>  ignore

RUN                                  OUTP1                                  RUN
    DDLM_RD_Outp_Upd.req(Outp_Data)
     \Outp_Data.len > 0
     =>  FDL_REPLY_UPDATE.req(SSAP=57, L_sdu=Outp_Data,
           Serv_Class=Low, Transmit=Multiple)

RUN                                  OUTP2                                  RUN
    DDLM_RD_Outp_Upd.req(Outp_Data)
     \Outp_Data.len = 0
     =>  ignore

RUN                                  OUTP3                                  RUN
    FDL_DATA_REPLY.ind(DSAP=57)
     =>  ignore

RUN                                  SPRM1                                  RUN
    FDL_DATA_REPLY.ind(DSAP=61)
     =>  DDLM_Set_Prm.ind(Req_Add=Loc_add, Prm_Data=L_sdu)

RUN                                  SCFG1                                  RUN
    FDL_DATA_REPLY.ind(DSAP=62)
     =>  DDLM_Chk_Cfg.ind(Req_Add=Loc_add, Cfg_Data=L_sdu)
```

RUN **GCFG1** **RUN**
DDLM_Get_Cfg_Upd.req(Real_Cfg_Data)
=> FDL_REPLY_UPDATE.req(SSAP=59, L_sdu=Real_Cfg_Data,
Serv_class=Low, Transmit=Multiple)

RUN **GCFG2** **RUN**
FDL_DATA_REPLY.ind(DSAP=59)
=> ignore

RUN **GCTR1** **RUN**
FDL_DATA.ind(DSAP=58)
\L_sdu.len = 2
=> DDLM_Global_Control.ind(Req_Add=Loc_add,
Control_Command=L_sdu[1], Group_Select=L_sdu[2])

RUN **GCTR2** **RUN**
FDL_DATA.ind(DSAP=58)
\L_sdu.len # 2
=> ignore

RUN **SADR1** **RUN**
FDL_DATA_REPLY.ind(DSAP=55)
\L_sdu.len = 4
=> DDLM_Set_Slave_Add.ind(New_Slave_Add=L_sdu[1],
Ident_Number=L_sdu[2-3], No_Add_Chg=L_sdu[4])

RUN **SADR2** **RUN**
FDL_DATA_REPLY.ind(DSAP=55)
\L_sdu.len > 4
=> DDLM_Set_Slave_Add.ind(New_Slave_Add=L_sdu[1],
Ident_Number=L_sdu[2-3], No_Add_Chg=L_sdu[4],
Rem_Slave_Data=L_sdu[5-L_sdu.len])

RUN **SADR3** **RUN**
FDL_DATA_REPLY.ind(DSAP=55)
\L_sdu.len < 4
=> ignore

RUN **RUN1** **RUN**
FDL_REPLY_UPDATE.con(OK)
=> ignore

RUN **RUN2** **RUN**
FMA1/2_EVENT.ind
=> ignore

RUN **ERR1** **RUN**
FDL_REPLY_UPDATE.con(LS/LR/IV)
=> DDLM_Fault.ind

RUN **ERR2** **RUN**
FDL_DATA_REPLY.ind
\0 <= DSAP <= 54
=> DDLM_Fault.ind

RUN **ERR3** **RUN**

FDL_DATA.ind
\DSAP # 58
=> DDLM_Fault.ind

RUN **ERR4** **RUN**

inadmissible FDL-primitive
=> DDLM_Fault.ind

RUN **ERR5** **RUN**

unknown FDL-primitive
=> DDLM_Fault.ind

14 Characteristic Features of a Device

14.1 General

The features described in the following section define the characteristic features and the performance of a PROFIBUS-DP device.

A vendor of DP-Slaves and DP-Masters (class 1) shall offer the Device data base (DDB) as a device data sheet and a device data file to the user. Using the Device data base enables checks of device data in the configuration phase of a PROFIBUS-DP system. This way, errors can be avoided during the configuration. Because of the defined file format of the Device data base, it is possible to realize vendor independent configuration tools for PROFIBUS-DP systems. The configuration tool uses the DDB for testing the data that were entered regarding the observance of limits and validity related to the performance of the individual device.
The distinction of the DDB data sets is achieved by the vendor- and device-identifiers.
In the case of composite-devices, PROFIBUS-FMS specific Device data base information is inserted directly at the beginning of the Device data base file. The section "Format of the Device Data Base File" fixes the format of the description of the device data for a PROFIBUS-DP device.

14.2 Format of the Device Data Base File

The Device data base file is an ASCII-file. It may be created with every applicable ASCII text editor. The DP-specific part begins always with the identifier "#Profibus_DP". The device data base will be specified respectively as parameter of a key word. At the evaluation of the key words the kind of letters, capital or small, will be don't care. The sequence of the parameters may be in any order. The data medium which the vendor of the DP-device uses for the delivery of the Device data base is not defined in this specification.
The file format is line oriented. Each line contains statements for exactly one parameter. If a semicolon is detected during the interpretation of the line, it is assumed that the rest of the line is a comment. The maximum number of characters per line is fixed to 80. If it not possible to describe the information in one line then it is allowed to use continuation lines. A "\" at the end of a line indicates that the following line is a continuation line. It is distinguished between number-parameters and text-parameters. No special end-identifier is defined. But it is to be ensured that the file ends after a complete line. Parameters which are not used for a DP-Master or a DP-Slave shall be omitted.

14.3 Meaning and Coding of the Key Words

14.3.1 General

The type-identifier which is specified with the key words refers to the parameter with the same name. The parameters are distinguished between:

- Mandatory (M)
- Optional (O)
- Optional, with default=0 if the parameter is not present (D)
- at least one out of the group (G) suitable for the corresponding baud rate

14.3.2 General DP-Key words

Vendor_Name: (M)
Name of the vendor of the DP-device.
Type: Visible String (32)

Model_Name: (M)
Controller-Type of the DP-device.
Type: Visible String (32)

Revision: (M)
Revision of the DP-device.
Type: Visible String (32)

Ident_Number: (M)
Type number of the DP-device (see section "Manufacturer Identifier").
Type: Unsigned16

Protocol_Ident: (M)
Protocol-identifier of the DP-device.
Type: Unsigned8
0: PROFIBUS-DP,
16 to 255: Vendor specific

Station_Type: (M)
Type of the DP-device.
Type: Unsigned8
0: DP-Slave,
1: DP-Master (class 1)

FMS_supp: (D)
This device is a FMS/DP-composite device.
Type: Boolean (1: True)

Hardware_Release: (M)
Hardware release of the DP-device.
Type: Visible String (32)

Software_Release (M)
Software release of the DP-device.
Type: Visible String (32)

9.6_supp: (G)
The DP-device supports the baud rate 9.6 kBaud.
Type: Boolean (1: True)

19.2_supp: (G)
The DP-device supports the baud rate 19.2 kBaud.
Type: Boolean (1: True)

93.75_supp: (G)
The DP-device supports the baud rate 93.75 kBaud.
Type: Boolean (1: True)

187.5_supp: (G)
The DP-device supports the baud rate 187.5 kBaud.
Type: Boolean (1: True)

500_supp: (G)
The DP-device supports the baud rate 500 kBaud.
Type: Boolean (1: True)

1.5M_supp: (G)
The DP-device supports the baud rate 1.5 MBaud.
Type: Boolean (1: True)

MaxTsdr_9.6: (G)
This is the time that a responder at the baud rate 9.6 kBaud requires at maximum to answer after a request has been received (see Part 4).
Type: Unsigned16
      Time base: bit time

MaxTsdr_19.2: (G)
This is the time that a responder at the baud rate 19.2 kBaud requires at maximum to answer after a request has been received (see Part 4).
Type: Unsigned16
      Time base: bit time

MaxTsdr_93.75: (G)
This is the time that a responder at the baud rate 93.75 kBaud requires at maximum to answer after a request has been received (see Part 4).
Type: Unsigned16
      Time base: bit time

MaxTsdr_187.5: (G)
This is the time that a responder at the baud rate 187.5 kBaud requires at maximum to answer after a request has been received (see Part 4).
Type: Unsigned16
      Time base: bit time

MaxTsdr_500: (G)
This is the time that a responder at the baud rate 500 kBaud requires at maximum to answer after a request has been received (see Part 4).
Type: Unsigned16
      Time base: bit time

MaxTsdr_1.5M: (G)
This is the time that a responder at the baud rate 1.5 MBaud requires at maximum to answer after a request has been received (see Part 4).
Type: Unsigned16
      Time base: bit time

Redundancy: (D)
This parameter specifies if the device supports redundant transmission technology.
Type: Boolean
0: not supported, 1: redundancy supported

Repeater_Ctrl_Sig: (D)
This parameter specifies the signal-type of the CNTR-P signal at the connector.
Type: Unsigned8
0: Not connected, 1: RS485, 2:TTL

24V_Pins: (D)
This parameter specifies the function of the signals M24V and P24V at the connector.
Type: Unsigned8
0: Not connected, 1:Input, 2:Output

14.3.3 DP-Master (Class 1) related Key words

Download_supp: (D)
The DP-device supports the functions Download, Start_seq and End_seq.
Type: Boolean (1: True)

Upload_supp: (D)
The DP-device supports the functions Upload, Start_seq and End_seq.
Type: Boolean (1: True)

Act_Para_Brct_supp: (D)
The DP-device supports the function Act_Para_Brct.
Type: Boolean (1: True)

Act_Param_supp: (D)
The DP-device supports the function Act_Param.
Type: Boolean (1: True)

Max_MPS_Length: (M)
Maximal size of memory (in Byte) that a DP-device provides for the storage of the master parameter set.
Type: Unsigned32

Max_Lsdu_MS: (M)
This parameter specifies the maximal L_sdu length for all master-slave communication relationships.
Type: Unsigned8

Max_Lsdu_MM: (M)
This parameter specifies the maximal L_sdu length for all master-master communication relationships.
Type: Unsigned8

Min_Poll_Timeout: (M)
This parameter specifies the maximal time interval that a DP-Master (class 1) needs for the execution of a master-master function.
Type: Unsigned16
Time base: 10 ms

Trdy_9.6: (G)
This parameter specifies the time interval at the baud rate 9.6 kBaud that a DP-Master (class 1) needs to be prepared to receive a reply frame after sending a request PDU (see Part 4).
Type: Unsigned8
Time base: bit time

Trdy_19.2: (G)
This parameter specifies the time interval at the baud rate 19.2 kBaud that a DP-Master (class 1) needs to be prepared to receive a reply frame after sending a request PDU (see Part 4).
Type: Unsigned8
Time base: bit time

Trdy_93.75: (G)
This parameter specifies the time interval at the baud rate 93.75 kBaud that a DP-Master (class 1) needs to be prepared to receive a reply frame after sending a request PDU (see Part 4).
Type: Unsigned8
Time base: bit time

Trdy_187.5: (G)
This parameter specifies the time interval at the baud rate 187.5 kBaud that a DP-Master (class 1) needs to be prepared to receive a reply frame after sending a request PDU (see Part 4).
Type: Unsigned8
Time base: bit time

Trdy_500: (G)
This parameter specifies the time interval at the baud rate 500 kBaud that a DP-Master (class 1) needs to be prepared to receive a reply frame after sending a request PDU (see Part 4).
Type: Unsigned8
Time base: bit time

Trdy_1.5M (G)
This parameter specifies the time interval at the baud rate 1.5 MBaud that a DP-Master (class 1) needs to be prepared to receive a reply frame after sending a request PDU (see Part 4).
Type: Unsigned8
Time base: bit time

Tqui_9.6: (G)
This parameter specifies the transmitter fall time (T_{QUI}) (see Part 4) at the baud rate 9.6 kBaud.
Type: Unsigned8
Time base: bit time

Tqui_19.2: (G)
This parameter specifies the transmitter fall time (T_{QUI}) (see Part 4) at the baud rate 19.2 kBaud.
Type: Unsigned8
Time base: bit time

Tqui_93.75: (G)
This parameter specifies the transmitter fall time (T_{QUI}) (see Part 4) at the baud rate 93.75 kBaud.
Type: Unsigned8
Time base: bit time

Tqui_187.5: (G)
This parameter specifies the transmitter fall time (T_{QUI}) (see Part 4) at the baud rate 187.5 kBaud.
Type: Unsigned8
Time base: bit time

Tqui_500: (G)
This parameter specifies the transmitter fall time (T_{QUI}) (see Part 4) at the baud rate 500 kBaud.
Type: Unsigned8
Time base: bit time

Tqui_1.5M: (G)
This parameter specifies the transmitter fall time (T_{QUI}) (see Part 4) at the baud rate 1.5 MBaud.
Type: Unsigned8
Time base: bit time

Tset_9.6: (G)
This parameter specifies the setup time relating to layer 2. This time specifies the interval between the arrival of an event and the corresponding reaction (see Part 4) at the baud rate 9.6 kBaud.
Type: Unsigned8
Time base: bit time

Tset_19.2: (G)
This parameter specifies the setup time relating to layer 2. This time specifies the interval between the arrival of an event and the corresponding reaction (see Part 4) at the baud rate 19.2 kBaud.
Type: Unsigned8
Time base: bit time

Tset_93.75: (G)
This parameter specifies the setup time relating to layer 2. This time specifies the interval between the arrival of an event and the corresponding reaction (see Part 4) at the baud rate 93.75 kBaud.
Type: Unsigned8
Time base: bit time

Tset_187.5: (G)
This parameter specifies the setup time relating to layer 2. This time specifies the interval between the arrival of an event and the corresponding reaction (see Part 4) at the baud rate 187.5 kBaud.
Type: Unsigned8
Time base: bit time

Tset_500: (G)
This parameter specifies the setup time relating to layer 2. This time specifies the interval between the arrival of an event and the corresponding reaction (see Part 4) at the baud rate 500 kBaud.
Type: Unsigned8
Time base: bit time

Tset_1.5M: (G)
This parameter specifies the setup time relating to layer 2. This time specifies the interval between the arrival of an event and the corresponding reaction (see Part 4) at the baud rate 1.5 MBaud.
Type: Unsigned8
     Time base: bit time

LAS_Len: (M)
This parameter specifies how many entries the device permits into the list of active stations (LAS).
Type: Unsigned8

Tsdi_9.6: (G)
This value specifies the station delay time (Tsdi) of the initiator (see Part 4) at the baud rate 9.6 kBaud.
Type: Unsigned16
     Time base: bit time

Tsdi_19.2: (G)
This value specifies the station delay time (Tsdi) of the initiator (see Part 4) at the baud rate 19.2 kBaud.
Type: Unsigned16
     Time base: bit time

Tsdi_93.75: (G)
This value specifies the station delay time (Tsdi) of the initiator (see Part 4) at the baud rate 93.75 kBaud.
Type: Unsigned16
     Time base: bit time

Tsdi_187.5: (G)
This value specifies the station delay time (Tsdi) of the initiator (see Part 4) at the baud rate 187.5 kBaud.
Type: Unsigned16
     Time base: bit time

Tsdi_500: (G)
This value specifies the station delay time (Tsdi) of the initiator (see Part 4) at the baud rate 500 kBaud.
Type: Unsigned16
     Time base: bit time

Tsdi_1.5M: (G)
This value specifies the station delay time (Tsdi) of the initiator (see Part 4) at the baud rate 1.5 MBaud.
Type: Unsigned16
     Time base: bit time

Max_Slaves_supp: (M)
This parameter specifies how many DP-Slaves a DP-Master (class 1) can handle.
Type: Unsigned8

14.3.4 DP-Slave related Key Words

Freeze_Mode_supp: (D)
The DP-device supports the Freeze-Mode. DP-Slaves which support the freeze mode has to ensure that already in the next data exchange cycle after the Freeze control command the last frozen values of the inputs have to be transferred.
Type: Boolean (1: True)

Sync_Mode_supp: (D)
The DP-device supports the Sync-Mode.
Type: Boolean (1: True)

Auto_Baud_supp: (D)
The DP-device supports automatic baud rate detection.
Type: Boolean (1: True)

Set_Slave_Add_supp: (D)
The DP-device supports the function Set_Slave_Add.
Type: Boolean (1: True)

User_Prm_Data_Len: (D)
This parameter specifies the maximal length of User_Prm_Data.
Type: Unsigned8

User_Prm_Data: (O)
Type: Octet-String
The meaning of this parameter is vendor dependent. It specifies the default value for the User_Prm_Data field (see section "Send Parameter Data").

Min_Slave_Intervall: (M)
This parameter specifies the time between two slave poll cycles for a DP-device which has at least to expire .
Type: Unsigned16
Time base: 100 µs

Modular_Station: (D)
This parameter specifies if the related DP-device is a modular device.
Type: Boolean (0: compact device, 1: modular device)

Max_Module: (M if Modular_Station)
This parameter specifies the maximum number of modules of a modular station.
Type: Unsigned8

Max_Input_Len: (M if Modular_Station)
This parameter specifies the maximum length of the input data (in bytes) of a modular station.
Type: Unsigned8

Max_Output_Len: (M if Modular_Station)
This parameter specifies the maximum length of the output data (in bytes) of a modular station.
Type: Unsigned8

Max_Data_Len: (M if Modular_Station)
This parameter specifies the maximum length of the input- and output data (the sum in bytes) of a modular station. If this parameter is omitted the maximum length is the sum of the input- and output data.
Type: Unsigned16

Unit_Diag_Bit: (O)
In order to display vendor dependent status- or error messages it is possible to assign a text (Diag_Text) to each bit of the device related diagnostic field (see section "Examples for DDB-File Entries").
Used parameters:
 Bit:
  Type: Unsigned16
  Meaning: Bit position in the device related diagnostic field.
            (LSB in the first Byte is bit 0).
 Diag_Text:
  Type: Visible-String(32)

Unit_Diag_Area: (O)
Between the key words Unit_Diag_Area and Unit_Diag_Area_End the assignment of values of the device related diagnostic field to text strings (Diag_Text) is specified in a bit field (see section "Examples for DDB-File Entries").
Used parameters:
 First_Bit:
  Type: Unsigned16
  Meaning: First bit position of the bit field.
              (LSB of the first byte is bit 0)
 Last_Bit:
  Type: Unsigned16
  Meaning: Last bit position of the bit field. The bit field may be
           maximum 16 bit wide.
 Value:
  Type: Unsigned16
  Meaning: Value in the bit field.

 Diag_Text:
  Type: Visible-String

Module: (M)
Between the key words Module and Endmodule it shall be specified either the identifier resp. identifiers of a DP-compact device, or the identifier resp. identifiers of all possible modules of a modular DP-Slave. Furthermore it is possible to specify vendor dependent error types in the channel related diagnostic field. If the key word Channel_Diag is used outside of the key words Module and Endmodule, it is specified that for all following modules the same vendor dependent error type in the channel related diagnostic field will be used (see section "Examples for DDB-File Entries").
Used parameters:
 Mod_Name:
  Type: Visible-String (32)
  Meaning: Module name of an individual module, which is used in a
           modular station, or device name of a compact DP-Slave.

 Config:
  Type: Octet-String (17)
  Meaning: This parameter specifies the identifier resp. identifiers
           of the module of a modular DP-Slave, or of a compact
           DP-device. The format of the identifier resp. identifiers
           is described in section "Check Configuration Data".

```
Channel_Diag: (O)
With the key word Channel_Diag the assignment of the vendor specific Error_Types
to text-strings (Diag_Text) in the channel related diagnostic field shall be
specified (see section "Examples for DDB-File Entries").
Used parameters:
 Error_Type:
  Type: Unsigned8  (16 <= Error_Type <= 31)
 Diag_Text:
  Type: Visible-String(32)
```

14.3.5 Formal Description of the DDB-File Format

The statements in square brackets are optional in the following description. The symbol "|" specifies the logical or-operation.

```
<WS>                  = <Space> | <Tab> | <WS><Space> | <WS><Tab>
<CRLF>                = <Carriage Return><Line Feed> |
                        <Carriage Return> | <Line Feed>
<Num>                 =  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<Namechar>            = a | b | c | d | e | f | g | h | i | j | k | l
                          | m | n | o | p | q | r | s | t | u | v | w | x
                      | y | z | _ | . | - | A | B | C | D | E | F | G
                      | H | I | J | K | L | M | N | O | P | Q | R | S
                      | T | U | V | W | X | Y | Z | <Num>
<Otherchar>           = + | * | / | < | > | ( | ) | [ | ] | { | } | !
                      | $ | % | & | ? | ´ | ^ | | | = | # | ; | , | :
                      | `
<Backslash>           = \
<Stringchar>          = <Namechar> | <Otherchar>
<Char>                = <Stringchar> | "
<Com>                 = ; | <Com><Char> | <Com><WS>
<ComLn>               = <Com><CRLF>
<LineEnd>             = <CRLF> | <Com><CRLF> | <WS><LineEnd> |
                        <LineEnd><ComLn>|<LineEnd><CRLF>
<Boolean>             = 0 | 1
<Decimal>             = <Num> | <Decimal><Num>
<Hexchar>             = <Num> | A | B | C | D | E | F | a | b | c | d
                      | e | f
<Hexadecimal>         = 0x<Hexchar> | <Hexadecimal><Hexchar>
<Number>              = <Decimal> | <Hexadecimal>
<Octet>               = <Number>  { 0 <= <Octet> <= 255 }
<Unsigned8>           = <Octet>
<Unsigned16>          = <Number>  { 0 <= <Unsigned16> <= 65535 }
<Unsigned32>          = <Number>  { 0 <= <Unsigned32> <= 4294967295 }
<Octet-String>        = <Octet> | <Octet-String>,<Octet>
<String>              = <Stringchar> | <Space>
                      | <String><Stringchar> | <String><Space>
<Visible-String>      = "<String>"
<Unit>                = Bit | us
<Key word>            = <Namechar> | <Key word><Namechar>
<Any-String>          = <Char> | <WS> | <Any-String><Char>
                      | <Any-String><WS>
<Any-Line>            = <Any-String><CRLF>
<Long-Line>           = <Any-String><Backslash><CRLF> | <Any-Line>
                      | <Long-Line>
<Any-Text>            = <Any-Line> | <Any-Text><Any-Line>
<User-Definition>     = <Key word>[<Otherchar><Any-Line>]<LineEnd>
<Vendor_Name>         = <Visible-String>     { Length <= 32 }
<Model_Name>          = <Visible-String>     { Length <= 32 }
<Revision>            = <Visible-String>     { Length <= 32 }
<Ident_Number>        = <Unsigned16>
```

```
<Protocol_Ident>        = <Unsigned8>
<Station_Type>          = <Unsigned8>
<FMS_supp>              = <Boolean>
<Hardware_Release>      = <Visible-String>      { Length <= 32 }
<Software_Release>      = <Visible-String>      { Length <= 32 }
<9.6_supp>              = <Boolean>
<19.2_supp>             = <Boolean>
<93.75_supp>            = <Boolean>
<187.5_supp>            = <Boolean>
<500_supp>              = <Boolean>
<1.5M_supp>             = <Boolean>
<MaxTsdr_9.6>           = <Unsigned16>
<MaxTsdr_19.2>          = <Unsigned16>
<MaxTsdr_93.75>         = <Unsigned16>
<MaxTsdr_187.5>         = <Unsigned16>
<MaxTsdr_500>           = <Unsigned16>
<MaxTsdr_1.5M>          = <Unsigned16>
<Redundancy>            = <Boolean>
<Repeater_Ctrl_Sig>     = <Unsigned8>
<24V_Pins>              = <Unsigned8>
<Download_supp>         = <Boolean>
<Upload_supp>           = <Boolean>
<Act_Para_Brct_supp>    = <Boolean>
<Act_Param_supp>        = <Boolean>
<Max_MPS_Length>        = <Unsigned32>
<Max_Lsdu_MM>           = <Unsigned8>
<Max_Lsdu_MS>           = <Unsigned8>
<Min_Poll_Timeout>      = <Unsigned16>
<Trdy_9.6>              = <Unsigned8>
<Trdy_19.2>             = <Unsigned8>
<Trdy_93.75>            = <Unsigned8>
<Trdy_187.5>            = <Unsigned8>
<Trdy_500>              = <Unsigned8>
<Trdy_1.5M>             = <Unsigned8>
<Tqui_9.6>              = <Unsigned8>
<Tqui_19.2>             = <Unsigned8>
<Tqui_93.75>            = <Unsigned8>
<Tqui_187.5>            = <Unsigned8>
<Tqui_500>              = <Unsigned8>
<Tqui_1.5M>             = <Unsigned8>
<Tset_9.6>              = <Unsigned8>
<Tset_19.2>             = <Unsigned8>
<Tset_93.75>            = <Unsigned8>
<Tset_187.5>            = <Unsigned8>
<Tset_500>              = <Unsigned8>
<Tset_1.5M>             = <Unsigned8>
<Tsdi_9.6>              = <Unsigned16>
<Tsdi_19.2>             = <Unsigned16>
<Tsdi_93.75>            = <Unsigned16>
<Tsdi_187.5>            = <Unsigned16>
<Tsdi_500>              = <Unsigned16>
<Tsdi_1.5M>             = <Unsigned16>
<LAS_Len>               = <Unsigned8>
<Max_Slaves_supp>       = <Unsigned8>
<Freeze_Mode_supp>      = <Boolean>
<Sync_Mode_supp>        = <Boolean>
<Set_Slave_Add_supp>     = <Boolean>
<Auto_Baud_supp>        = <Boolean>
<User_Prm_Data_Len>     = <Unsigned8>
<User_Prm_Data>         = <Octet-String>
<User_Prm_Data_Def>     = <Octet-String>
<Min_Slave_Intervall>   = <Unsigned16>
<Modular_Station>       = <Boolean>
```

```
<Max_Module>            = <Unsigned8>
<Max_Input_Len>         = <Unsigned8>
<Max_Output_Len>        = <Unsigned8>
<Max_Data_Len>          = <Unsigned16>
<Bit>                   = <Unsigned16>
<Diag_Text>             = <Visible-String>
<First_Bit>             = <Unsigned16>    { 0 <= <First-Bit> <= 510 }
<Last_Bit>              = <Unsigned16>    { 1 <= <Last-Bit> <= 511 }
<Value>                 = <Unsigned16>
<Mod_Name>              = <Visible-String>
<Config>                = <Octet-String>
<Error_Type>            = <Unsigned8>     { 16 <= <Error_Type> <= 31 }
<Value-Item>            =
    Value(<Value>)[<WS>]=[<WS>]<Diag_Text><LineEnd>
<Value_List>            = <Value_Item> | <Value-List><Value-Item>
<Unit-Diag-Area-Def>    =
    Unit_Diag_Area[<WS>]=[<WS>]<First-Bit>-<Last-Bit><LineEnd>
    <Value-List>
    Unit_Diag_Area_End<LineEnd>
<Channel-Diag-Definition> =
    Channel_Diag(<Value>)[<WS>]=[<WS>]<Diag_Text><LineEnd>
<Unit-Def-Item>         =
    Vendor_Name[<WS>]=[<WS>]<Vendor_Name><LineEnd>
  | Model_Name[<WS>]=[<WS>]<Model_Name><LineEnd>
  | Revision[<WS>]=[<WS>]<Revision><LineEnd>
  | Ident_Number[<WS>]=[<WS>]<Ident_Number><LineEnd>
  | Protocol_Ident[<WS>]=[<WS>]<Protocol_Ident><LineEnd>
  | Station_Type[<WS>]=[<WS>]<Station_Type><LineEnd>
  | Hardware_Release[<WS>]=[<WS>]<Hardware-Release><LineEnd>
  | Software_Release[<WS>]=[<WS>]<Software-Release><LineEnd>
  | 9.6_supp[<WS>]=[<WS>]<9.6_supp><LineEnd>
  | 19.2_supp[<WS>]=[<WS>]<19.2_supp><LineEnd>
  | 93.75_supp[<WS>]=[<WS>]<93.75_supp><LineEnd>
  | 187.5_supp[<WS>]=[<WS>]<187.5_supp><LineEnd>
  | 500_supp[<WS>]=[<WS>]<500_supp><LineEnd>
  | 1.5M_supp[<WS>]=[<WS>]<1.5M_supp><LineEnd>
  | MaxTsdr_9.6[<WS>]=[<WS>]<MaxTsdr_9.6><LineEnd>
  | MaxTsdr_19.2[<WS>]=[<WS>]<MaxTsdr_19.2><LineEnd>
  | MaxTsdr_93.75[<WS>]=[<WS>]<MaxTsdr_93.75><LineEnd>
  | MaxTsdr_187.5[<WS>]=[<WS>]<MaxTsdr_187.5><LineEnd>
  | MaxTsdr_500[<WS>]=[<WS>]<MaxTsdr_500><LineEnd>
  | MaxTsdr_1.5M[<WS>]=[<WS>]<MaxTsdr_1.5M><LineEnd>
  | Redundancy[<WS>]=[<WS>]<Redundancy><LineEnd>
  | Repeater_Ctrl_Sig[<WS>]=[<WS>]<Repeater_Ctrl_Sig><LineEnd>
  | 24V_Pins[<WS>]=[<WS>]<24V_Pins><LineEnd>
  | Download_supp[<WS>]=[<WS>]<Download_supp><LineEnd>
  | Upload_supp[<WS>]=[<WS>]<Upload_supp><LineEnd>
  | Act_Para_Brct_supp[<WS>]=[<WS>]<Act_Para_Brct_supp><LineEnd>
  | Act_Param_supp[<WS>]=[<WS>]<Act_Param_supp><LineEnd>
  | Max_MPS_Length[<WS>]=[<WS>]<Max_MPS_Length><LineEnd>
  | Max_Lsdu_MM[<WS>]=[<WS>]<Max_Lsdu_MM><LineEnd>
  | Max_Lsdu_MS[<WS>]=[<WS>]<Max_Lsdu_MS><LineEnd>
  | Min_Poll_Timeout[<WS>]=[<WS>]<Min_Poll_Timeout><LineEnd>
  | Trdy_9.6[<WS>]=[<WS>]<Trdy_9.6><LineEnd>
  | Trdy_19.2[<WS>]=[<WS>]<Trdy_19.2><LineEnd>
  | Trdy_93.75[<WS>]=[<WS>]<Trdy_93.75><LineEnd>
  | Trdy_187.5[<WS>]=[<WS>]<Trdy_187.5><LineEnd>
  | Trdy_500[<WS>]=[<WS>]<Trdy_500><LineEnd>
  | Trdy_1.5M[<WS>]=[<WS>]<Trdy_1.5M><LineEnd>
  | Tqui_9.6[<WS>]=[<WS>]<Tqui_9.6><LineEnd>
  | Tqui_19.2[<WS>]=[<WS>]<Tqui_19.2><LineEnd>
  | Tqui_93.75[<WS>]=[<WS>]<Tqui_93.75><LineEnd>
  | Tqui_187.5[<WS>]=[<WS>]<Tqui_187.5><LineEnd>
```

```
  | Tqui_500[<WS>]=[<WS>]<Tqui_500><LineEnd>
  | Tqui_1.5M[<WS>]=[<WS>]<Tqui_1.5M><LineEnd>
  | Tset_9.6[<WS>]=[<WS>]<Tset_9.6><LineEnd>
  | Tset_19.2[<WS>]=[<WS>]<Tset_19.2><LineEnd>
  | Tset_93.75[<WS>]=[<WS>]<Tset_93.75><LineEnd>
  | Tset_187.5[<WS>]=[<WS>]<Tset_187.5><LineEnd>
  | Tset_500[<WS>]=[<WS>]<Tset_500><LineEnd>
  | Tset_1.5M[<WS>]=[<WS>]<Tset_1.5M><LineEnd>
  | Tsdi_9.6[<WS>]=[<WS>]<Tsdi_9.6><LineEnd>
  | Tsdi_19.2[<WS>]=[<WS>]<Tsdi_19.2><LineEnd>
  | Tsdi_93.75[<WS>]=[<WS>]<Tsdi_93.75><LineEnd>
  | Tsdi_187.5[<WS>]=[<WS>]<Tsdi_187.5><LineEnd>
  | Tsdi_500[<WS>]=[<WS>]<Tsdi_500><LineEnd>
  | Tsdi_1.5M[<WS>]=[<WS>]<Tsdi_1.5M><LineEnd>
  | LAS_Len[<WS>]=[<WS>]<LAS_Len><LineEnd>
  | Max_Slaves_supp[<WS>]=[<WS>]<Max_Slaves_supp><LineEnd>
  | Freeze_Mode_supp[<WS>]=[<WS>]<Freeze_Mode_supp><LineEnd>
  | Sync_Mode_supp[<WS>]=[<WS>]<Sync_Mode_supp><LineEnd>
  | Auto_Baud_supp[<WS>]=[<WS>]<Auto_Baud_supp><LineEnd>
  | Set_Slave_Add_supp[<WS>]=[<WS>]<Set_Slave_Add_supp><LineEnd>
  | User_Prm_Data_Len[<WS>]=[<WS>]<User_Prm_Data_Len><LineEnd>
  | User_Prm_Data[<WS>]=[<WS>]<User_Prm_Data><LineEnd>
  | User_Prm_Data_Def[<WS>]=[<WS>]<User_Prm_Data_Def><LineEnd>
  | Min_Slave_Intervall[<WS>]=[<WS>]<Min_Slave_Intervall><LineEnd>
  | Modular_Station[<WS>]=[<WS>]<Modular_Station><LineEnd>
  | Max_Module[<WS>]=[<WS>]<Max_Module><LineEnd>
  | Max_Input_Len[<WS>]=[<WS>]<Max_Input_Len><LineEnd>
  | Max_Output_Len[<WS>]=[<WS>]<Max_Output_Len><LineEnd>
  | Max_Data_Len[<WS>]=[<WS>]<Max_Data_Len><LineEnd>
  | Unit_Diag_Bit(<Bit>)[<WS>]=[<WS>]<Diag_Text><LineEnd>
  | <Unit-Diag-Area-Def>
  | <Channel-Diag-Definition>
  | <User-Definition>
<Unit-Definition-List>  =
    <Unit-Def-Item> | <Unit-Definition-List><Unit-Def-Item>
<Module-Def-Item> =
    <Channel-Diag-Definition>
  | <User-Definition>
<Module-Def-List> =
    <Module-Def-Item> | <Module-Def-List><Module-Def-Item>
<Module-Definition>     =
    Module[<WS>]=[<WS>]<Mod_Name><WS><Config><LineEnd>
    <Module-Def-List>
    EndModule<LineEnd>
<Module-Definition-List>      =
    <Module-Definition> | <Module-Definition-List><Module-Definition>
<GSD>             =
    [<Any-Text>]
    #Profibus_DP<LineEnd>
    <Unit-Definition-List>
    <Module-Definition-List>
    [#<Key word><LineEnd>[<Any-Text>]]
```

14.3.6 Examples for DDB-File Entries

Example 1:

```
#Profibus_DP
Vendor_Name      = "Tretter,Weber,Szabo,Schweigert"
Model_Name       = "Emmerling,Volz,Thiesmeier"
Revision         = "3.0 Hindelang"
Protocol_Ident   = 0
Station_Type     = 0
Hardware_Release = "V1.0"
Software_Release = "V1.1"
Trdy_1.5M        = 11; Master is recipient after 11 bit times
Modular_Station  = 1; modular DP-Slave
Unit_Diag_Bit(0)    = "Error on Input 1"
Unit_Diag_Bit(1)    = "Error on Input 2"
; the text "Error on Input 1" is assigned to the LSB (Bit0) of the
; device related diagnostic field.
; the text "Error on Input 2" is assigned to the Bit 1 of the
; device related diagnostic field.
;
Unit_Diag_Area = 1-5
Value(1) = "Error on Input 1 and 2"
Value(2) = "Power failure"
Value(3) = "Valve 1 out of function"
Unit_Diag_Area_End
; A bit field from bit 1 until bit 5 is created in the device
; related diagnostic field. Error messages are issued via this bit
; field. If, as an Example the value 2 (x,x,0,0,0,1,0,x) is issued,
; the error message "Power failure" is displayed on a
; centralized display.
;
Module = "Input module 16I-GT" 0x11
Channel_Diag(16) = "Overtemperature or Overload"
Channel_Diag(17) = "Cable broken or short circuit"
EndModule
Module = "Output module 32O-0.5A" 0x23
Channel_Diag(16) = "Overtemperature or Overload"
Channel_Diag(17) = "Cable broken or short circuit"
EndModule
;
; At a modular DP-Slave two different modules are applicable.
; Module one is an input module with 16 bits with the name
; "Input module 16I-GT" and the identifier 11H. For this module a
; channel related diagnostic, and a vendor specific error
; type 17 "Cable broken or short circuit", and a error type 16
; "Over temperature or Overload" exist.
; Module 2 is an output module with 32 bits with the name
; "Output module 32O-0.5A" and the identifier 23H. For the
; channel related diagnostic of this module, a vendor specific
; error type 17 "Cable broken or short circuit" and the error
; type 16 "Over temperature or Overload" exist.
```

Example 2:

```
Unit_Diag_Area = 0-5
Value(0) = "No error"
Value(1) = "Error on Input 0 - 23"
Value(2) = "Error on Output 0 - 15"
Value(3) = "24V failed"
Unit_Diag_Area_End
;
```

```
Module = "DP compact device 24I/16A" 0x12,0x21
EndModul
; Here, a compact DP-device is described with the name
; "DP compact device 24I/16A" with the identifiers 12H, 21H.
; In the device related diagnostic field a bit field 0 to 5 is
; defined, through which 4 status/error messages may be handled.
```

15 Application Characteristics

15.1 Restrictions

In the PROFIBUS-DP system, the maximal PDU size is restricted to 246 bytes as specified in the Data Link Layer Service Protocol Specifications. Preferably, the PDU size should not exceed 32 bytes.
This substantiates on the components that have been available at specification time of this specification. For future applications it is possible to use the maximum possible PDU size of 246 byte.

15.2 Time Behaviour

The following diagram shows the theoretical system reaction time (see section "System Reaction Time") of a DP-Master (class 1) in dependance of the connected amount of DP-Slaves with 2 bytes input- and output-data each.
It is presumed that all configured DP-Slaves are in the user data exchange mode with no diagnostic message pending. The DP-Slaves answer immediately after the minimum Station Delay Timer (min T_{SDR} = 11 t_{Bit}) is expired. At the DP-Master (class 1) T_{SDI} is assumed as 37 bit times.

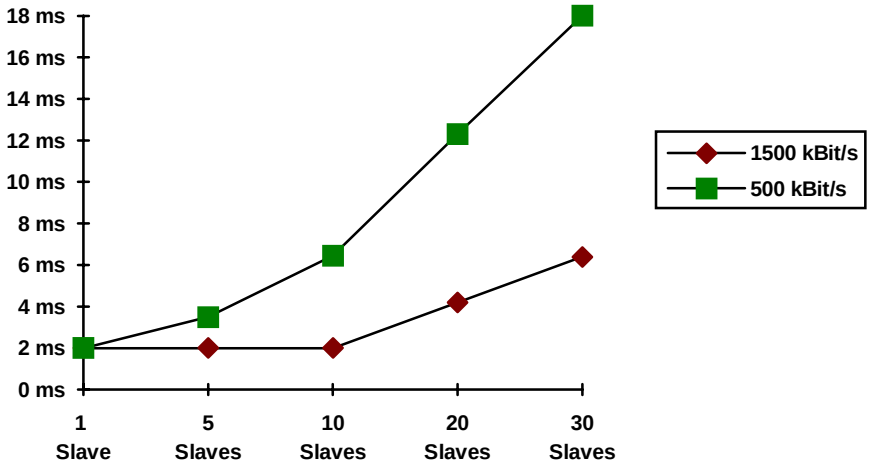


Figure 42. Cycle time of the DP-system

15.3 Manufacturer Identifier

The manufacturer identifier (Ident_Number) is necessary for all DP-devices except for DP-Master (class 2). For each type of device an individual manufacturer identifier will be needed. This manufacturer identifier is not a series number. If a manufacturer have got a manufacturer identifier for one type of device it is allowed to him to use this number for all produced devices of the same type. At devices of the same type but with differences in the amount of inputs and outputs it is possible to use the same manufacturer identifier for each produced DP-device. Assumption: The device can be described in the DDB as a modular device.
Each device type shall have an individual Ident_Number. This Ident_Number will be issued by the PNO (PROFIBUS Nutzerorganisation e.V.).

Annex 2-B (normative)

Mixed Operation

2-B.1 Mixed Operation of FMS- and DP-Devices on the same Line

The mixed operation of both device types on the same line is possible. In the mixed mode, the FMS-devices communicate with each other and the DP-devices communicate with each other. A communication between application instances of FMS- and DP-devices on the same line is not possible. For both device types, the medium access is realized according to the the Data Link Layer Service Protocol Specifications.

During the configuration of a mixed system, the choice of the layer 2 bus parameters shall be considered carefully. The following conditions are valid:

- Each master shall be able to accept a Ready-Time $T_{RDY} \leq 250\ t_{Bit}$.
- After power-on, FMS-devices shall operate with the default-values which are specified for the FMS-Operation-Mode.
- After power-on, DP-devices shall operate with the bus parameters which are specified in table 2-B.1. In this operation mode no special precautions are necessary in the DP-Slave.
- The configuration of the DP-Slaves and FMS-devices shall be harmonized one after another (station address, bus parameter, etc.).
- FMS-devices shall not use the SAPs 54 to 62 and NIL.

Table 2-B.1. Default-Bus Parameters for DP-devices in mixed operation with FMS-devices which only support baud rates up to 500kBit/s.

Baud rate in kbit/s	9.6	19.2	93.75	187.5	500
Default-DP-Master					
T_{SL} (t_{Bit})	125	250	600	1500	3500
min T_{SDR} (t_{Bit})	30	60	125	250	500
max T_{SDR} (t_{Bit})	60	120	250	500	1000
T_{SET} (t_{Bit})	1	1	1	1	1
T_{QUI} (t_{Bit})	0	0	0	0	0
G	1	1	1	1	1
HSA	126	126	126	126	126
max_retry_limit	1	1	1	1	1
Default-DP-Slave					
min T_{SDR} (t_{Bit})	11	11	11	11	11

An optimization of the bus parameters may enhance the data transfer rate. In this case, the optimized bus parameters shall be loaded into the FMS- and into the DP-devices.

2-B.2 Mixed Operation of DP and FMS in the same Device

2-B.2.1 General

For special applications it may be necessary to realize devices that contain both PROFIBUS layer 7 protocol (FMS) and the PROFIBUS-DP protocol in a single implementation. This possibility exists because both PROFIBUS variations uniquely use the FDL and FMA1/2-services for the medium access protocol.

The principle protocol architecture is shown in figure 2-B.1.

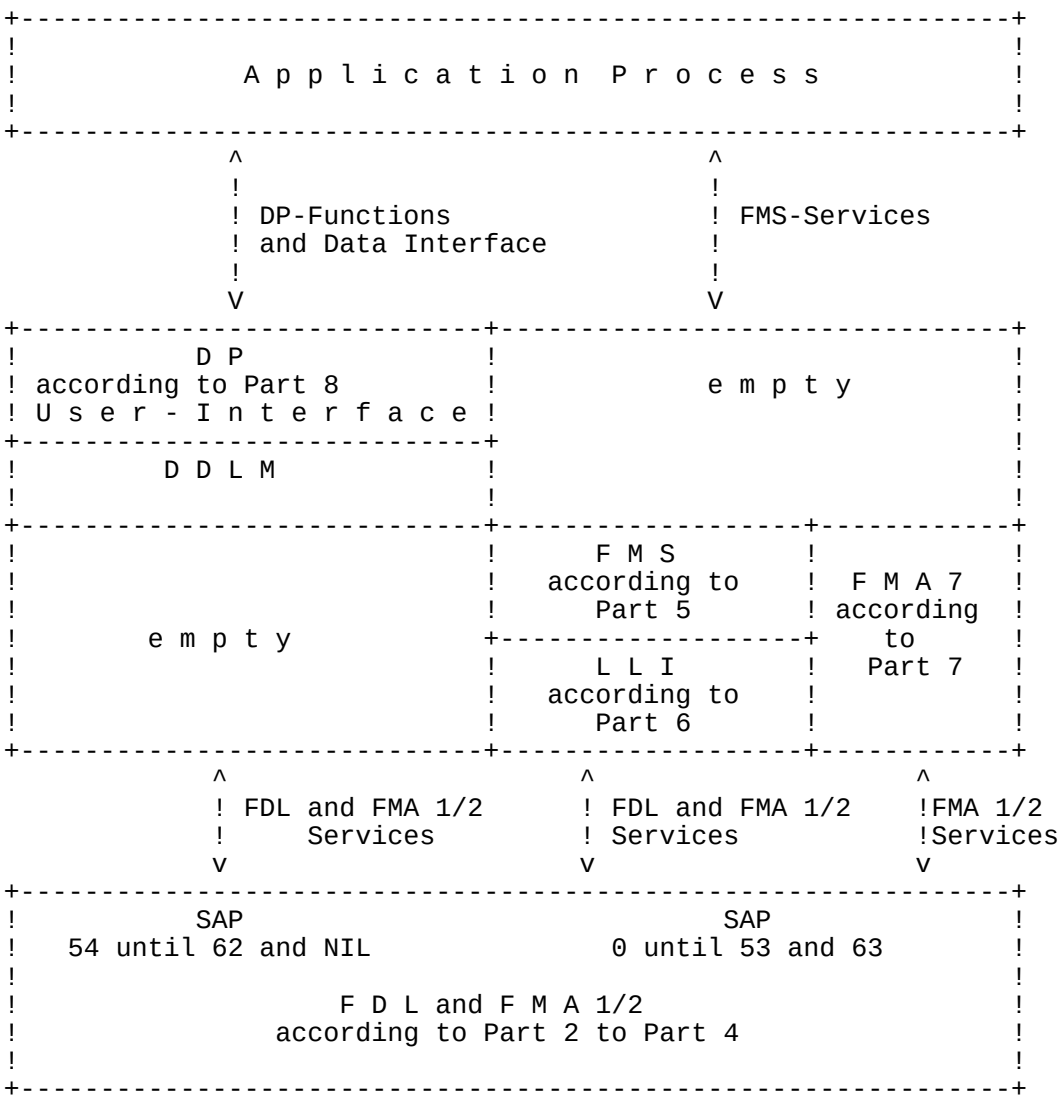


Figure 2-B.1. Architecture of composite PROFIBUS Implementations

In this protocol architecture the FDL takes care of the distinction between a PROFIBUS-DP function or a FMS-service. This distinction is based on the used service access points. Both protocols use the FDL service access points as a sub-addressing mechanism.

In the case of PROFIBUS-DP, fixed functions are assigned to the service access points. By addressing a certain service access point, a fixed function is called implicitly.
In the case of PROFIBUS-FMS, the service access points are used for the assignment of the layer 2 L_sdu´s to the logical connections.

Composite devices are able to communicate on application level with other FMS-devices using FMS-functions and to communicate with other DP-devices using DP-functions. Composite devices are able to communicate in any way with other FMS- or DP-devices on the same line.

During the configuration and operation of these devices, the choice of the layer 2 bus parameters shall be handled with special observance.

In a DP-system, a composite device starts with DP-default bus parameters. In a mixed system, it starts with the default values defined in table 2-B.2.

For a composite device the station address shall not be changed over the bus (Set_Slave_Add). In case of changing the bus parameters during the operation mode (FMA7 functions or Set_Bus_Par) it shall be ensured that the bus parameters are changed in both protocol-instances.

In a composite device a local management is necessary for the co-ordination of the parallel execution of the FMA1/2-services. This management is not taken into consideration in this specification.

At start-up or reset of a composite device, the FMA7 co-ordinates the reset functions of both protocol-instances.

2-B.2.2 Configuration Instructions for Composite Devices

If in the same device PROFIBUS-FMS-services and PROFIBUS-DP-functions are simultaneously executed, the following configuration-instructions shall be noticed:

At the FMS-Slave / DP-Slave:

- The service access points 55 until 62 and NIL are reserved for the DP-protocol.

- All other service access points (0 until 54 and 63) are usable for the FMS-protocol without restrictions.

At the FMS-master / DP-Master:

- The service access points 54, 62 and NIL are reserved for the DP-protocol.

- All other service access points are usable without restrictions for the FMS-protocol.

In both cases, the instructions of section "Bus Parameter" shall be noticed for the choice of the layer 2 bus parameters.

Table 2-B.2. Default Bus Parameter for slave composite device in mixed operation with FMS- and DP-devices.

Baud rate in kbit/s	9.6	19.2	93.75	187.5	500	1500
min T_{SDR} (t_{Bit})	30	60	125	250	255	255

For the calculation of the system reaction time in the mixed mode, the device specific features for the execution of layer 2 shall be noticed. All DP-Master - DP-Slave functions are executed with high priority.
Therefore there are two alternatives to prioritize DP- and FMS-services:

a) DP-functions should be preferred against FMS-services.

- FMS-services should executed with low priority.

- The choice of the Target Rotation Time should be made that way that at each token receipt a complete slave poll cycle can be executed, see section "Token Rotation Time".

- At each token receipt only one FMS-Service should be executed.

b) DP- and FMS-services shall have the same priority.

- The choice of the Target Rotation Time should be made that way that at each token receipt a complete slave poll cycle and in addition several FMS-services should be executed.

- The Target Rotation Time shall have the same value in all masters.

PROFIBUS Specification - Normative Parts

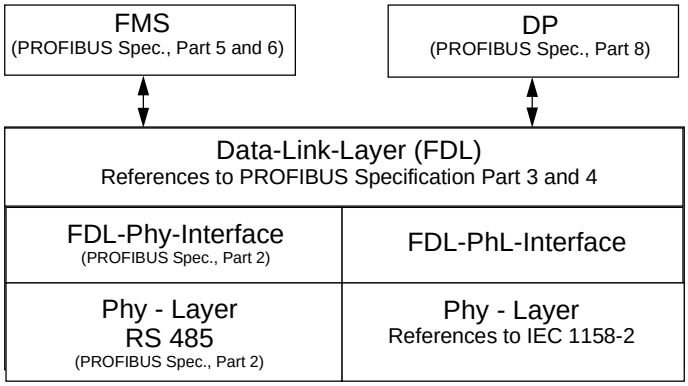
Part 9

Physical Layer and Data Link Layer for Process Automation

Contents

1 Introduction

This part describes, as already announced in the PROFIBUS Specification, Part 2 Data Transmission, a further data transmission technique, consisting of the transmission medium (Physical Medium) and the accompanying Physical Layer. The connecting Data Link Layer and the management of the Physical Layer comply to the PROFIBUS Specification, except for the changes and additions described in this part.



FMS : Fieldbus Message Specification
DP : Decentralized Periphery
FDL : Fieldbus Data Link Layer

Figure 1: Relationship of this part to the other parts

The interfaces to the Application Layer (FDL and FMA1/2) are specified as in PROFIBUS Specification Layer 2 Interface.

The logical interface used between PhL and FDL is as defined in PROFIBUS Specification "Data Transmission". The protocol for the bus line, access and transmission complies to PROFIBUS Specification "Medium Access Methods" and "Transmission Protocol".

The data transmission is defined according to IEC 1158-2 which enables in addition the intrinsic safe variant in the protection class "Intrinsic Safe" and power supply over the bus.

Regarding the Medium Attachment Unit (MAU), the following transmission rate defined in IEC 1158-2, clause 11, is used:

- 31.25 kbit/s, voltage mode and wire medium.

The management assigned to the Application Layer and described in Part 7 shall take the Physical Layer variables of IEC 1158-2 and their associated ranges of values into account (see subclause 10.4 of this part).

Applications in process automation (e.g. process industries) require a simple transmission medium (2 wire cable) in different topologies, such as line and tree. Furthermore, low power and low cost bus connections with real time behavior, i.e. with guaranteed response time, are necessary. Severe electromagnetic interference and explosive atmospheres may have to be dealt with.

This specification shall contribute to the low cost interconnection of digital field devices from different vendors in a distributed fieldbus system as well as towards ensuring reliable communication.

2 Field of Application

This specification defines functional, electrical and mechanical features of a serial fieldbus system for applications in process automation (e.g. process industries). The data transmission defined in this specification is the Physical Layer ’version 1’ (31.25 kbit/s, voltage mode, wire medium) of IEC 1158-2.

3 Normative References

This specification makes dated and undated references to specifications in other publications. These normative references are located at the respective points in the text and the publications are named accordingly. In case of fixed references, later changes or revised versions of these publications only belong to this specification if they are worked in through changes or revised versions. For undated references the last edition is valid.

ISO 7498 : 1984	Information processing systems; Open Systems Interconnection; Basic Reference Model
IEC 1158-2 :1993	Fieldbus standard for use in industrial control systems - Part 2: Physical Layer specification and service definition
DIN EN 50020	Electrical apparatus for potentially explosive atmospheres - intrinsic safety "i"

4 General

4.1 Definitions

The definitions of the PROFIBUS Specification and of IEC 1158-2 apply.

4.2 Abbreviations

Note that the Physical Layer is no longer abbreviated as ”PHY”, but as ”PhL” in accordance with IEC 1158-2. ”Ph-” is used as the prefix for service primitives.

Abbreviations used :

CRC	Cyclic Redundancy Check
DCE	Data Communication Equipment
DTE	Data Terminal Equipment
MAU	Medium Attachment Unit
MDS	Medium Dependent Sublayer
PhICI	Physical Layer Interface Control Information
Ph-	Physical-
PhL	Physical Layer
PhID	Physical Layer Interface Data
PhIDU	Physical Layer Interface Data Unit

PhPCI	Physical Layer Protocol Control Information
SDF	Start Delimiter Data Link
SDL 1	Start Delimiter 1 Data Link
SDL 2	Start Delimiter 2 Data Link
SDL 3	Start Delimiter 3 Data Link
SDL 4	Start Delimiter 4 Data Link
SDL 5	Start Delimiter 5 Data Link
T_{PTG}	Post-transmission gap time

Refer also to IEC 1158-2 clause 4 as well as PROFIBUS Specification, Part 2, subclause 3.1.2.

4.3 Basic Properties

The same basic features shall apply as described in PROFIBUS Specification, Part 2, subclause 3.2.

5 Characteristic Features

Beside the requirements of the application field explained in subclause 3.3 of PROFIBUS Specification, Part 2, this specification has to enable intrinsically safe data transmission and power supply over the bus. This results in the characteristic features described as follows:

Network topology:	Linear bus, terminated at both ends, with or without stubs and branches (tree)
Medium, distances, number of stations:	according to IEC 1158-2, subclause 11.2.2, rules 1 to 3, the following values shall apply: - non-intrinsically safe fieldbus with and without power supply: 2 to 32 devices - intrinsically safe fieldbus with power supply: limitations on the number of stations is the result of the limited electrical power that intrinsically safe circuits are able to transfer. - The distance between two stations, when the maximum number of stations is connected, is at most 1.9 km including the stubs.
Transmission speed:	31.25 kbit/s for distances up to 1,900 m
Redundancy:	with second medium
Transmission characteristics:	Half duplex, synchronous, self-clocking, Manchester-Biphase-L-Coding
Addressing, station types, bus access, data transfer services, frame length, data integrity:	according to PROFIBUS Specification, Part 2, subclause 3.3

6 Scope

The part of the specification is based on the architectural model, the bus access protocol and the transmission procedures as described in the PROFIBUS Specification "Data Link Layer".

The data transmission technique of Layer 1 (Physical Layer, PhL) is based on the rules defined in IEC 1158-2, clause 11.

The frame format builds on the synchronous Protocol Data Unit (Ph-PDU) described in clause 9 of IEC 1158-2. It consist of a Preamble, a Start Delimiter and an End Delimiter as well as the Layer 2 data (FDL-PDU). The data are coded and decoded according to the Manchester-Biphase-L-Code.

The transmission protocol of Layer 2 as well as the Layer 1 management (FMA1) are defined according to IEC 1158-2, clauses 5 and 6.

Appendix A explains possible structures for repeaters, fieldbus interfaces and systems with several fieldbus lines connected to a single master station. Guidelines on a redundant central control unit and a bus analyzer/diagnostic unit are added, too. Finally, the number of connectable stations for intrinsic safe fieldbus with power supply over the bus as well as the data transfer rate and the system reaction time are calculated for an example under consideration of the frame formats that have been changed by the integration of IEC 1158-2.

7 Data Transmission (Physical Medium, Physical Layer)

7.1 Electrical Characteristics

The Physical Layer described in IEC 1158-2 clause 11 shall be used.

Version 1: 31.25 kbit/s, voltage mode and wire medium (intrinsically safe Physical Layer)

Repeater

An extension of line length and an increase in the number of stations may be achieved by means of bi-directional amplifiers (called repeaters, see clause A.1 of this part). The maximum number of repeaters between two stations is four. The following maximum values shall apply for 31.25 kbit/s and if the bus segments are in series (linear bus topology):

- 31.25 kbit/s:

  1 Repeater: 3.8 km and 62 stations

  2 Repeater: 5.7 km and 92 stations (see figure 2)

  3 Repeater: 7.6 km and 122 stations

  4 Repeater: 9.5 km and 127 stations

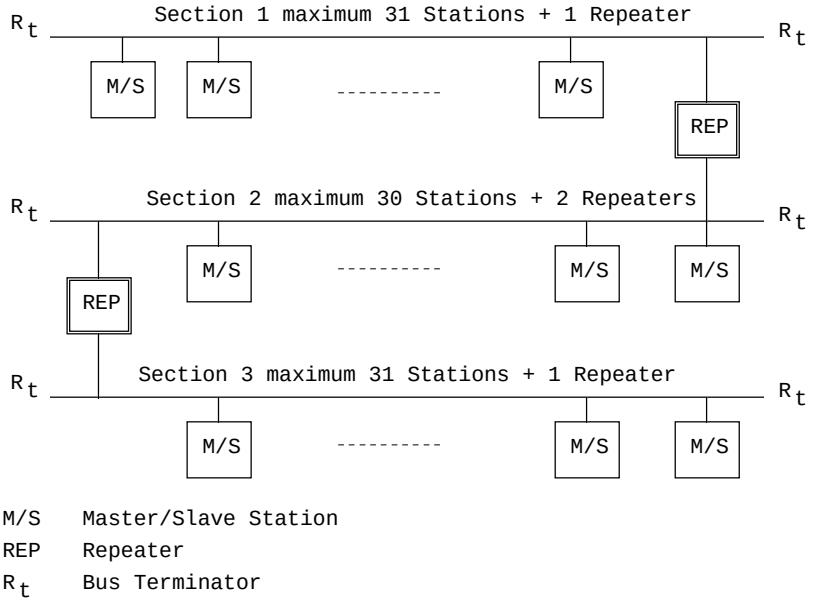


Figure 2: Repeater in Linear Bus Topology (cf Fig. 2 of Part 2)

For a tree topology of segments the following maximum values shall apply at 31.25 kbit/s:

e.g.. 5 Repeaters: 7.6 km and 127 stations

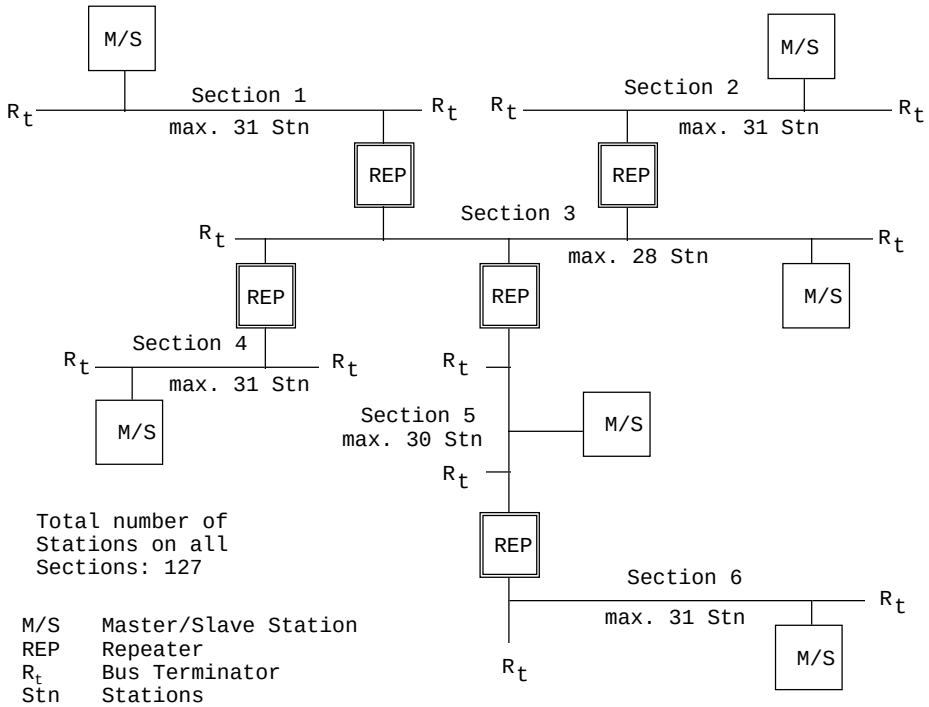


Figure 3: Repeater in Tree Topology (cf Fig. 3 of Part 2)

7.2 Interface between Physical Layer (PhL) and Medium Access and Transmission Protocol (FDL)

This clause gives an abstract definition of the Ph-Data service that is provided to the FDL-Layer by the Physical Layer. The Ph-Data service is for the receipt and transmission of data (one octet at a time).

The execution of the interface and the control within a station are not fixed or stipulated.

The following service primitives have been taken unchanged from IEC 1158-2 and have been repeated for clarity.

Ph-DATA request (class, data)

Ph-DATA indication (class, data)

Ph-DATA confirmation (status)

The parameter **class** specifies the Physical Layer Interface Control Information (PhICI) of the Physical Layer Interface Data Unit (PhIDU). The following values are possible for the Ph-Data request primitive:

a) START-OF-ACTIVITY

b) DATA

c) END-OF-DATA-AND-ACTIVITY

For the Ph-DATA indication primitive the parameter class contains the following values:

a) START-OF-ACTIVITY

b) DATA

c) END-OF-DATA

d) END-OF-ACTIVITY

e) END-OF-DATA-AND-ACTIVITY

They are available to the interface between physical medium and medium access control protocol and transmission protocol.

The parameter **data** specifies the Physical Layer Interface Data (PhID) of the Physical Layer Interface Data Unit (PhIDU). It consists of an octet Physical Layer User Data which has to be sent (request) or received (indication).

The parameter **status** marks either the success or a detected failure of the transmission.

The Ph-DATA confirm primitive marks the end of a transmission. It is passed on to the FDL-controller after a transmission has been completed and it indicates thereby the Physical Layer’s readiness to receive the next Ph-Data request primitive.

The Physical Layer Data service with primitives as well as the definition of the parameter values are described in IEC 1158-2, clause 5.

7.3 Redundancy of Physical Layer and Medium (optional)

To increase the reliability, a redundant transmission line is allowed by PROFIBUS "Data Transmission". Its configuration shall conform to the specifications stipulated in IEC 1158-2, subclause 11.2.2, rule 9.

In principle, the telegrams have to be sent simultaneously by several transmitters (usually more than two). A receiver is selected upon each reception. The parameter setting of the transmitter channels and the receiver channels is carried out by means of the Ph-SETVALUE request via the Physical Management Interface.

The selection of the appropriate receiver channel is carried out by Layer 2 (FDL) by watching over activity on the transmission lines independently of other stations. As described in Part 2, subclause 4.3, the following main criteria apply for switching from a given receiver channel:

- Two or more invalid frames are received in succession.

  Invalid means: invalid format and invalid CRC.

- Time-out T_{TO} expired, see subclause 8.1.7 of this part.

- No Syn Time T_{SYN} was notified during a Synchronization Interval Time T_{SYNI}, see subclause 8.1.7 of this part.

Dependent on the execution, further switch conditions may be selected.

8 Medium Access Methods and Transmission Protocol (Data Link Layer, FDL)

The medium access method and the transmission protocol (Data Link Layer, FDL), which are described in the PROFIBUS Specification "Medium Access Methods" and "Transmission Protocol", shall apply for this part, except for the deviations specified in the following corresponding subclauses. Due to the extensions to the frame formats (IEC 1158-2 synchronous transmission protocol), the timer calculations have to be modified in part as described in subclause 8.1.7 of this part.

8.1 Transmission Procedures and FDL Controller

8.1.1 Token Procedures

The specifications of subclause 4.1.1 of Part 4 shall apply.

8.1.1.1 Token Passing

The specifications of subclause 4.1.1.1 of Part 4 shall apply .

8.1.1.2 Addition and Removal of Stations

The specifications of subclause 4.1.1.2 of Part 4 shall apply

8.1.1.3 (Re)Initialization the Logical Token Ring

The specifications of subclause 4.1.1.3 of Part 4 shall apply

8.1.1.4 Token Rotation Time

The specifications of subclause 4.1.1.4 of Part 4 shall apply.

8.1.1.5 Message Priorities

The specifications of subclause 4.1.1.5 of Part 4 shall apply.

8.1.2 Acyclic Request or Send / Request Mode

The specifications of subclause 4.1.2 of Part 4 shall apply.

8.1.3 Cyclic Send / Request Mode

The specifications of subclause 4.1.3 of Part 4 shall apply.

8.1.4 Request FDL Status of all Stations (Live List)

The specifications of subclause 4.1.4 of Part 4 shall apply.

8.1.5 Status of the FDL Controller

The specifications of subclause 4.1.5 of Part 4 shall apply.

8.1.6 FDL Initialization

The specifications of subclause 4.1.6 of Part 4 shall apply for FDL initialization. Because of the different Ph Layer complying to IEC 1158-2, the operational parameter T_{QUI} (Transmitter fall Time / Repeater switch time) shall be renamed post-transmission gap.

8.1.7 Timer Operation

The following times T are measured in bits as in the PROFIBUS Specification "Medium Access Methods" and "Transmission Protocol". A time t in seconds (s) shall therefore be divided by the Bit Time t_{BIT}.

Bit-Time t_{BIT}:

The Bit Time t_{bit} is the time which elapses during the transmission of one bit. It is equivalent to the reciprocal value of the transmission rate:

t_{BIT} =1 / Transmission Rate (bit/s) (1)

Syn Time T_{SYN}:

The minimum time interval during which each station shall receive no activity from the transmission medium before it may accept the begin of a request (request or send/request) or token frame.

The Syn Time corresponds to the post-transmission gap time that is defined in IEC 1158-2, clause 9. It is at least 4 bit and may be increased by FMA1/2 up to 32 bit.

$\mathbf{T_{SYN}}$ = T_{PTG} = Post-transmission Gap Time = T_{QUI}

$\mathbf{T_{SYN}}$ = 4 to 32 bit (2)

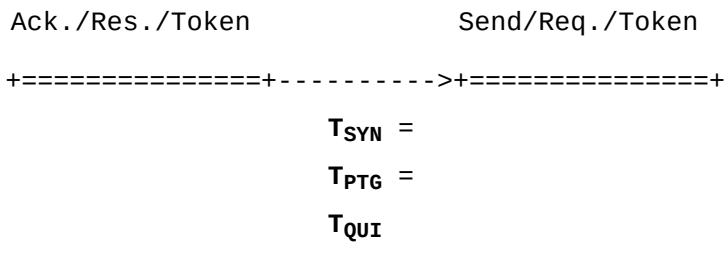


Figure 4: Syn-Time $\mathbf{T_{SYN}}$

Syn Interval Time $\mathbf{T_{SYNI}}$:

The Synchronization Interval Time T_{SYNI} serves to monitor the maximum allowed time interval between two consecutive receptions of Ph-DATA indication primitives of the classes: START-OF-ACTIVITY and END-OF-ACTIVITY (or END-OF-DATA-AND-ACTIVITY), in order to detect "permanent transmitters".

This time comprises two complete message sequences, each of which consists of two PDUs of maximum length and the associated maximum Physical Layer Control Information (PhPCI: Preamble, Start Delimiter, End Delimiter) and the maximum Syn Time (post-transmission gap).

Three Ph-Data indications of class END-OF-ACTIVITY (or END-OF-DATA-AND-ACTIVITY) may fail thereby.

$\mathbf{T_{SYNI}} = 2 \cdot (2 \cdot (T_{SYN} + Ph_{PCI} + T_{FDL})) + 64^{*)}$ (3)

*) The constant 64 constitutes a safety margin.

T_{FDL} = maximum length of a FDL frame (see clauses 8.5 and 8.6 of this part)

$T_{SYNI} = 2 \cdot (2 \cdot (32 + 80 + 255 \cdot 8))+64 = 8\ 672$ bit (4)

Station Delay Time $\mathbf{T_{SDx}}$:

The Station Delay Time T_{SDX} is the period of time which may elapse between a Ph-DATA request primitive (class: END-OF-DATA-AND-ACTIVITY) or Ph-DATA indication primitive (class: END-OF-ACTIVITY or END-OF-DATA-AND-ACTIVITY) until the following Ph-DATA request primitive (class: START-OF-ACTIVITY) or a Ph-DATA indication primitive (class: START-OF-ACTIVITY) (relative to the transmission medium, i.e. including the Physical Layer Entity).

The following three station delays are defined:

1) Station Delay of Initiator (station transmitting request or token frame)

$\mathbf{T_{SDI}} = t_{SDI} / t_{BIT}$ (5)

2) Minimum Station Delay of Responders (station that acknowledges or responds)

min $\mathbf{T_{SDR}}$ = min t_{SDR} / t_{BIT} (6)

3) Maximum Station Delay of Responders

$$\mathbf{max\ T_{SDR}} = \max t_{SDR} / t_{BIT} \quad (7)$$

Quiet Time T_{QUI} = T_{PTG}:

The Transmitter Fall Time or Repeater Switch Time corresponds to the Post-transmission Gap Time (T_{SYN}). The following shall apply:

$$\mathbf{T_{QUI}} = T_{PTG} = T_{SYN} \quad (8)$$

Ready Time T_{RDY}:

The Ready Time T_{RDY} is the time within which a master station shall be ready to receive an acknowledgment or response after transmitting a request. The Ready Time is defined as follows:

$$\mathbf{T_{RDY}} < \min T_{SDR} \quad (9)$$

In order to fulfill this condition it may be necessary to prolong T_{SDR}.

While disconnecting the transmitter the Quiet Time shall be considered. The readiness may start only after this time:

$$\mathbf{T_{QUI}} = T_{PTG} = T_{SYN} < T_{RDY} \quad (10)$$

In order to fulfill this condition the T_{SDR} shall be increased according to equation (9) if necessary.

Safety Margin T_{SM}:

The following time interval is defined as Safety Margin T_{SM}:

$$\mathbf{T_{SM}} = 2\ bit + 2 \cdot T_{SET} \quad (11)$$

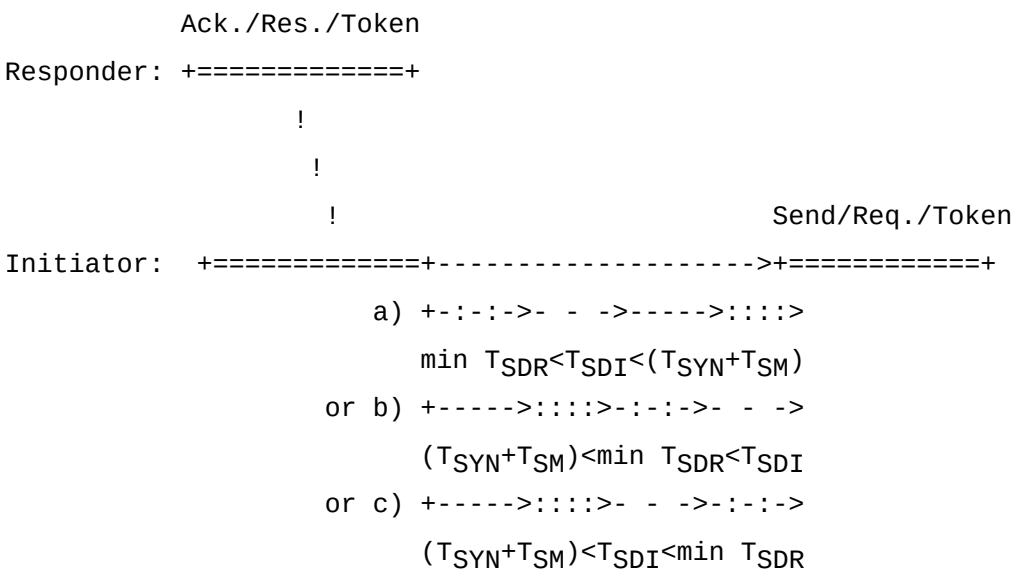
T_{SET} is the set-up time which expires from the occurrence of an event (e.g. Ph-Data confirm) until the execution of the necessary reaction:

$$\mathbf{T_{SET}} = t_{SET} / t_{BIT} \quad (12)$$

Idle Time T_{ID}:

The Idle Time T_{ID} is the time which expires at the initiator after a Ph-DATA indication primitive (class: END-OF-ACTIVITY or END-OF-DATA-AND-ACTIVITY) until reception of a new frame with Ph-DATA request primitive (class: START-OF-ACTIVITY) or after passing a Ph-DATA request primitive (class: END-OF-DATA-AND-ACTIVITY) with a Ph-DATA confirm primitive to transmit a frame which is not to be acknowledged until passing a new Ph-DATA request primitive (class: START-OF-ACTIVITY) for transmitting the next frame. The Idle Time shall be at least the Syn Time plus the Safety Margin T_{SM} (see description of Idle Time in subclause 4.1.7 of Part 4).

After an acknowledgment, response or token frame the Idle Time is defined as follows:



$$\mathbf{T_{ID1}} = \max (T_{SYN} + T_{SM}, \min T_{SDR}, T_{SDI}) \quad (13)$$

(The maximum value shall be used in each case)

Figure 5: Idle Time T_{ID1} (cf Fig. 3 of Part 4)

After a request frame, which is not to be acknowledged, the following Idle Time shall apply (cf also Fig. 4 of Part 4):

$$\mathbf{T_{ID2}} = \max (T_{SYN} + T_{SM}, \max T_{SDR}) \quad (14)$$

(The maximum value shall be used in each case)

Transmission Delay Time T_{TD}:

The Transmission Delay Time T_{TD} is the maximum time which elapses on the transmission medium between transmitter and receiver when a frame is transmitted. Delay times of repeaters shall be considered if necessary. The Transmission Delay Time is defined as follows:

$$\mathbf{T_{TD}} = t_{TD} / t_{BIT} \quad (15)$$

Compliant to IEC 1158-2, clause 11, rule 4, the value of 20 bit times shall not be surpassed.

Slot Time T_{SL}:

The Slot Time T_{SL} is the maximum time the initiator waits after passing a Ph-DATA request primitive (class: END-OF-DATA-AND-ACTIVITY) for transmitting a request frame from the Ph-DATA confirm primitive until receiving the first Ph-DATA indication primitive (class: DATA) as an indication of receiving the immediate acknowledgment or answer. Furthermore, T_{SL} is the maximum time the initiator waits for a Ph-DATA indication primitive (class: DATA) after the token frame as reaction to receiving a frame from the token receiver. Theoretically two Slot Times are distinguished (see description of Slot Time in subclause 4.1.7 of the Part 4, Fig. 5 and Fig. 6). After a request frame (request or send/request) the following Slot Time shall apply:

$$\mathbf{T_{SL1}} = 2 \cdot T_{TD} + \max T_{SDR} + T_{PRE} + 16 \text{ bit} + T_{SM} \quad (16)$$

After a token frame the following Slot Time shall apply:

$$\mathbf{T_{SL2}} = 2 \cdot T_{TD} + \max T_{ID1} + T_{PRE} + 16 \text{ bit} + T_{SM} \quad (17)$$

T_{PRE}: Preamble period (see IEC 1158-2)

Note: In order to simplify the realization, only the longer Slot Time is used in the system. This does not influence the system reaction time negatively, as the Slot Time is merely a monitoring time.

$$\mathbf{T_{SL}} = \max (T_{SL1}, T_{SL2}) \quad (18)$$

(The maximum value shall be used in each case)

Time-out T_{TO}:

The time-out T_{TO} serves to monitor the master and slave stations' line activity and Idle Time. Monitoring starts either immediately after the PON, in the "Listen_Token" or "Passive_Idle" state or later after the reception of a Ph-DATA indication primitive (class: END-OF-ACTIVITY or END-OF-DATA-AND-ACTIVITY). It finishes upon receipt of a Ph-DATA indication primitive (class: START-OF-ACTIVITY) for reception of a following frame. If the Idle Time reaches the Time-out value, the bus is regarded as inactive (error case, e.g. due to lost token). The Time-out is defined as follows (cf also Part 4, subclause 4.1.7 "Time-out"):

$$\mathbf{T_{TO}} = 6 \cdot T_{SL} + 2 \cdot n \cdot T_{SL} \quad (19)$$

For master stations: n = station address (0 to 126)
For slave stations: n = 130, independent of their station address

GAP Update Time T_{GUD}:

The specifications of the subclause 4.1.7 of Part 4 shall apply.

8.2 Cycle and System Reaction Times

8.2.1 Token Cycle Time

Similar to PROFIBUS Specification "Medium Access Methods" and "Transmission Protocol", the following shall apply for the Token Cycle Time T_{TC}:

$$\mathbf{T_{TC}} = T_{TF} + T_{TD} + T_{ID} \quad (20)$$

Due to the frame character (cf subclause 9.5.1 of this part) and the changed frame format (cf subclauses 8.6 and 8.7), the Token Frame Time T_{TF} is 64 bit with a preamble of 8 bit.

Furthermore, it shall be noted that only the transmission speed 31.25 kbit/s may be selected.

8.2.2 Message Cycle Time

The following shall apply for the Message Cycle Time T_{MC} :

$$\mathbf{T_{MC}} = T_{S/R} + T_{SDR} + T_{A/R} + T_{ID} + 2 \cdot T_{TD} \qquad (21)$$

The specifications for the Message Cycle Time stipulated in the subclause 4.2.2 of the Part 4 shall apply except for the following cases:

The PDU transmission times ($T_{S/R}$, $T_{A/R}$) are determined by the number of PDU octets.

It follows from this that:

$T_{S/R}$ = a · 8 bit a = number of octets in Send/Request PDU
$T_{A/R}$ = b · 8 bit b = number of octets in Ack/Response PDU

Example:

a = 9 for the Request PDU (1 octet Preamble):
$T_{S/R}$ =72 bit
b = 62 for the Response PDU (1 octet Preamble, 50 octet DATA_UNIT):
$T_{A/R}$ = 496 bit

8.2.3 System Reaction Times

The specifications for the System Reaction Times stipulated in the subclause 4.2.3 of Part 4, shall apply.

8.3 Error Control Procedures

As described in Part 4, subclause 4.3, errors in the line protocol (see IEC 1158-2, clause 9) and in the medium access protocol, such as erroneous start octets and CRC octets, frame length, response times etc. may result in the specified station reactions.

8.4 Timers and Counters

As explained in subclause 4.4 of Part 4, the following timers are needed to measure the Token Rotation Time and to realize the monitor timers:

Token Rotation Timer, Idle Timer, Slot Timer, Time-out Timer, Syn Interval Timer and GAP Update Timer.

Token Rotation Timer: The functionality of this timer is as defined in PROFIBUS Specification "Medium Access Methods" and "Transmission Protocol".

Idle Timer: This timer monitors the idle state $T_{PTG} = T_{SYN} = T_{QUI}$ on the bus line. The Idle Timer in the master station with the token is loaded with T_{ID1} or T_{ID2} depending on the data transmission service (cf subclause 9.1.7 of this part). The timer is decremented every bit time, when after a Ph-DATA request primitive (class: END-OF-DATA-AND-ACTIVITY) either the Ph-DATA confirm primitive at the transmitter or the Ph-DATA indication primitive (class: END-OF-ACTIVITY or END-OF-DATA-AND-ACTIVITY) at the receiver is transferred. A new request or a new token frame may be transmitted only after expiration of the timer.

Slot Timer: After a request from or a token transfer by a master station, this timer of the master station monitors whether the receiving station responds or becomes active within the defined Slot Time T_{SL}. The timer is initialized with

T_{SL} and is decremented every bit time after each transmission of a frame. This frame transmission is indicated by a Ph-DATA confirm primitive after transfer of a Ph-DATA request primitive (class: END-OF-DATA-AND-ACTIVITY). If the timer expires before a frame has been received, as indicated by a Ph-DATA indication primitive (class: START-OF-ACTIVITY), an error has occurred. As a result of that a retry or a new message cycle is initiated.

Time-out Timer: This timer serves to monitor the bus activity at master and slave stations. After transfer of a Ph-DATA request primitive (class: END-OF-DATA-AND-ACTIVITY) and return of a Ph-DATA confirm primitive or after receiving a Ph-DATA indication primitive (class: END-OF-ACTIVITY or END-OF-DATA-AND-ACTIVITY), the timer is loaded with a multiple of the Slot Time (cf subclause 9.1.7 of this part) and is decremented every bit time as long as no Ph-DATA indication primitive (class: START-OF-ACTIVITY) has been received. If the timer expires, a fatal error has occurred, which for the master station causes a (re)initialization. The FMA1/2 User of the slave or master station respectively receives a time-out notification (cf subclause 5.2.3.3 of Part 3).

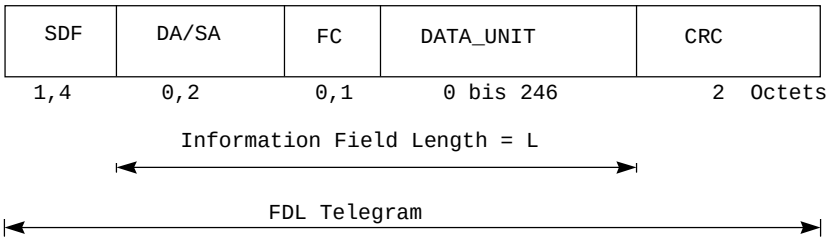
Syn Interval Timer: Master and slave stations use this timer to monitor the transmission medium for „permanent transmitters“. After every Ph-DATA indication primitive (class: START-OF-ACTIVITY) the timer is loaded with the value T_{SYNI} (cf subclause 9.1.7 of this part) and decremented every bit time, as long as no Ph-DATA indication primitive (class: END-OF-ACTIVITY or END-OF-DATA-AND-ACTIVITY) has been received. If the timer expires, an error of the transmission medium has occurred. The FMA1/2 User receives a corresponding notification (cf subclause 5.2.3.3 of Part 3).

GAP Update Timer: This timer operates in the same way as described in subclause 4.4 of Part 4.

The specifications stipulated in subclause 4.4 of Part 4 shall apply for the optional counters.

8.5 Frame Structure

Each FDL frame (FDL PDU) consists of a Start Delimiter Data Link, an Information Field and a Cyclic Redundancy Check (CRC). The Information Field is divided into an Address Field and a Control Field. Additionally, a Data Field may exist. The Information Field is Null in the Short-Acknowledgment.



Notation herein:

SDF[1]	Start Delimiter Data Link, Length 1 or 4 octets
DA	Destination Address - Information Field
SA	Source Address - Information Field
FC	Frame Control - Information Field
DATA_UNIT	Data Field, Length (L-3), maximum 246 octets
CRC	Cyclic Redundancy Check, Length 2 octets

Figure 6: FDL frame format

8.5.1 Frame Character

The Start Delimiter Data Link, the Information Field and the CRC consist of a number of octets of 8 bit each. Each FDL octet is structured as follows:

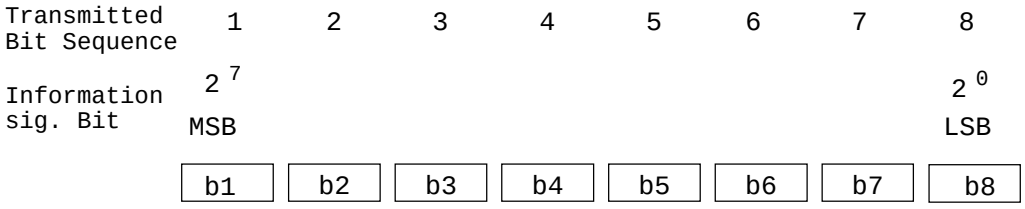


Figure 7: Octet Structure (cf Fig. 9 of Part 4, subclause 4.5.1)

8.6 Frame Formats

Based on the specifications of subclause 4.6 of Part 4 and its subclauses the different frame formats are described according to the frame format explained in subclause 9.5 of this part.

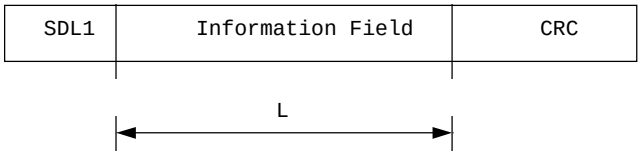
The figures contained in the following clauses do not show any sequences (request --> acknowledgment or response) but frame formats of the same category (Hd = 4; fixed length with/without data field and variable length), i.e. request frames may be followed by different acknowledgment or response frames (cf subclause 8.9 of this part).

---

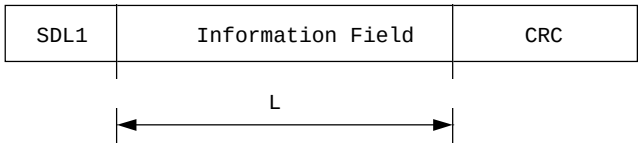
[1] The Start Delimiter Data Link may not be confused with the Start Delimiter SD of the Physical Layer.

8.6.1 Frames of fixed Length with no Data Field

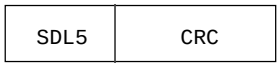
A)Request Frame



B)Acknowledgement Frame



C)Short Acknowledgement Frame



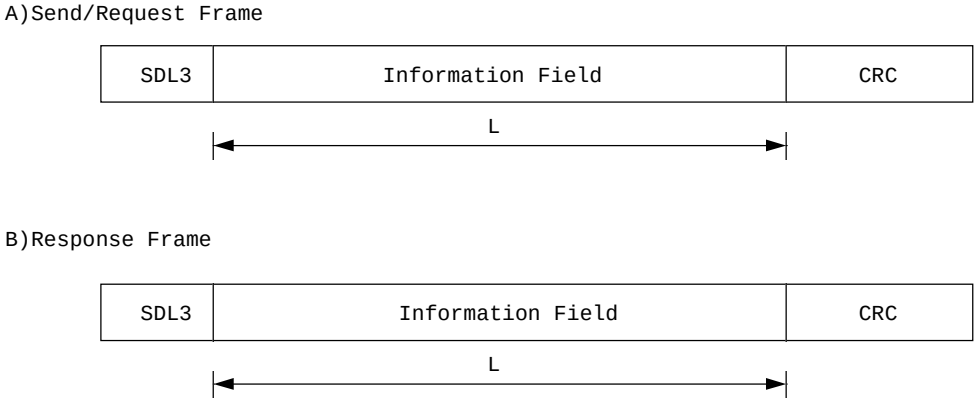
Notation herein:
SDL1 Start octet 1 (Start Delimiter 1 Data Link), Code: 10H
SDL5 Start octet 5 (Start Delimiter 5 Data Link), Code: E5H
CRC Cyclic Redundancy Check, 2 octets
L Information Field Length, fixed number of octets: L = 3
Information Field cf Part 4, subclause 4.6.1

Figure 8: Frames of fixed Length with no Data Field (cf Fig. 10 of Part 4)

Transmission Rules

Beside the transmission rules of the Ph Layer stipulated in IEC 1158-2, clause 9, the receiver shall check the DA/SA- and CRC octets for each frame. If the check produces a negative result the whole frame shall be discarded.

9.6.2 Frames of fixed Length with Data Field



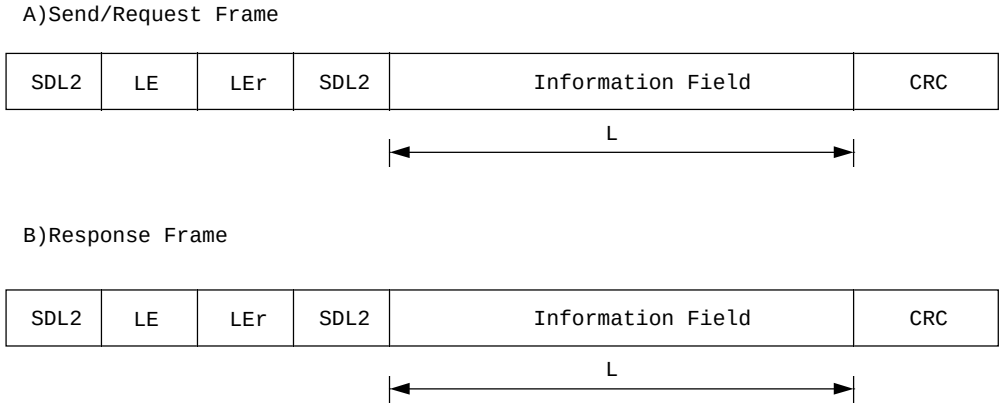
Notation herein:
SDL3 Start Delimiter 3 Data Link, Code: A2H
CRC Cyclic Redundancy Check, 2 octets
L Information Field Length, fixed number of octets: L = 11
Information Field cf Part 4, subclause 4.6.2

Figure 9: Frames of fixed Length with Data Field (cf Fig. 11 of Part 4)

Transmission Rules

The same transmission rules shall apply as for frames of fixed length with no Data Field (cf subclause 9.6.1 of this part)

8.6.3 Frames with variable Data Field Length



Notation herein:
SDL2 Start Delimiter 2 Data Link, Code: 68H
LE Length, Value: 4 to 249
LEr Length (repeated)
CRC Cyclic Redundancy Check, 2 octets
L Information Field Length, variable number of octets: L = 4 to 249
Information Field cf Part 4, subclause 4.6.3

Figure 10: Frames with variable Data Field Length (cf Fig. 12 of part 4)

Transmission Rules

The same transmission rules as for frames of fixed length with no data field shall apply (cf subclause 9.6.1 of this part). In addition, the receiver shall verify if LE and Ler coincide. The information octets shall be counted from the Destination Address (DA) up to the Cycling Redundancy Check and shall be compared with LE.

8.7 Token Frame

Notation herein:
SDL4 Start Delimiter 4 Data Link, Code: DCH
DA Destination Address
SA Source Address
CRC Cyclic Redundancy Check, 2 octets

Figure 11: Token Frame (cf Fig. 13 of Part 4)

Transmission Rules

The same transmission rules as for frames of fixed length with no data shall apply (cf subclause 9.6.1 of this part).

8.8 Length, Address, Control and Check Octet

8.8.1 Length Octet (LE)

The length octet described in subclause 4.7.1 of Part 4 shall apply.

8.8.2 Address Octet (DA/SA)

The address octet described in subclause 4.7.2 of Part 4 shall apply.

8.8.2.1 Link Service Access Point (LSAP)

The Link Service Access Point (LSAP) described in subclause 4.7.2.2 of Part 4 shall apply.

8.8.3 Control Octet (FC)

The control octet described in subclause 4.7.3 of Part 4 shall apply.

8.8.4 Check Octet (FCS)

In contrast to PROFIBUS Specification "Medium Access Methods" and "Transmission Protocol" a Cyclic Redundancy Check (CRC) is stipulated for the frames to be transmitted. The CRC is realized by calculating and appending a check field of 16 bit (2 octets). The calculation and analysis of the check field is realized as follows:

on the transmitting side

the message to be transmitted (without CRC), the polynomial necessary for calculating the CRC and the composite message (including CRC) shall be considered as vectors M(X), F(X) and D(X) of dimensions k, n-k and n respectively.

The message vector M(X) shall be defined to be

$$M(X) = m_1X^{k-1} + m_2X^{k-2} + \ldots + m_{k-1}X^1 + m_k \quad (22)$$

and the check field F(X) shall be defined to be:

$$F(X) = f_{n-k-1}X^{n-k-1} + \ldots + f_0 \quad (23)$$
$$= f_{15}X^{15} + \ldots + f_0$$

The composite vector D(X) for the complete message shall be constructed from the message and the CRC vector and shall be defined to be:

$$D(X) = M(X)X^{n-k} + F(X) \quad (24)$$
$$= m_1X^{n-1}+m_2X^{n-2}+ \ldots +m_kX^{n-k}+f_{n-k-1}X^{n-k-1}+ \ldots +f_0$$
$$= m_1X^{n-1} + m_2X^{n-2} + \ldots + m_kX^{16}+f_{15}X^{15}+ \ldots +f_0$$

The check field is the remainder upon division of F(X) by the generator polynomial G(X). The remainder is calculated as follows:

$$F(X) = L(X)(X^k+1) + M(X)X^{n-k} \pmod{G(X)} \quad (25)$$

where G(X) is the generator polynomial defined as:

$$G(X) = X^{n-k} + g_{n-k-1}X^{n-k-1} + \ldots + 1 \quad (26)$$
$$= X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^6 + X^3 + X^2 + X + 1$$

and L(X) is the maximal weight polynomial (all 1s) defined as:

$$L(X) = \frac{X^{n-k}+1}{X+1} = X^{n-k-1} + X^{n-k-2} + \ldots + X + 1 \quad (27)$$
$$= X^{15} + X^{14} + X^{13} + X^{12} + \ldots + X^2 + X + 1$$

The Cyclic Redundancy Check defined by means of the generator polynomial G(X) enables a Hamming Distance Hd = 4 for message lengths shorter than 344 octets and Hd = 5 for lengths shorter than 15 octets.

on the receiver side:

The received message consists of an octet sequence that is composed of the actual message and the check field. It shall be considered as a vector V(X) of dimension u :

$$V(X) = v_1X^{u-1} + v_2X^{u-2} + \ldots + v_{u-1}X + v_u \quad (28)$$

The remainder R(X) of V(X) is computed in the same way as for the transmission side:

$$R(X) = L(X)X^u + V(X)X^{n-k} \pmod{G(X)} \quad (29)$$

$$= r_{n-k-1}X^{n-k-1} + \ldots + r_0$$

If no error has occurred during message transmission, R(X) is equal to a constant remainder polynomial given by:

$$R_{ok}(X) = L(X)X^{n-k} \pmod{G(X)} \quad (30)$$

$$= X^{15} + X^{14} + X^{13} + X^9 + X^8 + X^7 + X^4 + X^2$$

The initial value of the remainder that has to be computed at the receiver side shall be the hexadecimal value 0FFFF.

8.8.5 Data Field (DATA_UNIT)

The data field described in subclause 4.7.5 of Part 4 shall apply.

8.9 Transmission Procedures

The transmission procedures described in subclause 4.8 of Part 4 shall apply.

9 PROFIBUS Layer 2 Interface

9.1 FDL User - FDL Interface

The FDL User - FDL Interface described in subclause 4.1 of Part 3 shall apply.

9.2 FMA1/2 User - FMA1/2 Interface

The FMA1/2 User - FMA1/2 Interface described in subclause 4.2 of Part 3 shall apply.

10 Management (FMA1/2)

10.1 General Description of FMA1/2 Functions

The general description of FMA1/2 functions described in subclause 5.1 of Part 3 shall apply.

10.2 FDL - FMA1/2 Interface

The FDL - FMA1/2 Interface described in subclause 5.2 of Part 3 shall apply.

10.2.1 Overview of Services

The services described in subclause 5.2.1 of Part 3 shall apply.

10.2.2 Overview of Interactions

The interactions described in subclause 5.2.2 of Part 3 shall apply.

10.2.3 Detailed Specification of Services and Interactions

The specification of services and interactions described in subclause 5.2.3 of Part 3 shall apply.

10.2.3.1 Reset FDL

The service described in subclause 5.2.3.1 of Part 3 Reset FDL shall apply.

10.2.3.2 Set Value FDL, Read Value FDL

Due to the integration of IEC 1158-2, the specifications of the services Set Value FDL and Read Value FDL, which are defined in the subclause 5.2.3.2 of Part 3, shall be extended with the following associated baud rate parameter (cf tables 28 and 29 of Part 3):

- Baud_rate: 31.25 kbit/s

10.2.3.3 Fault FDL

The specifications of the service Fault FDL defined in subclause 5.2.3.3 of Part 3 shall apply.

10.3 PhL - FMA1/2 Interface

The interface described in subclause 5.3 of Part 3 shall apply to describe the interface between PhL and FMA1/2.

10.3.1 Overview of Services

The Ph Layer, as described in IEC 1158-2, clause 6, provides the following services to FAM1/2

- Reset PhL

- Set Value PhL

- Get Value PhL

- Event PhL

The services shall have the same meaning as described in subclause 5.3.1 of Part 3. The Read Value PHY corresponds to the Get Value PhL.

10.3.2 Overview of Interactions

The following service primitives shall apply at the PhL - FMA1/2 Interface:

Service	Primitive	permissible for the following users
Reset PhL	Ph-RESET request	Master and Slave
Set Value PhL	Ph-SETVALUE request	Master and Slave
	Ph-SETVALUE confirm	Master and Slave
Get Value PhL	Ph-GETVALUE request	Master and Slave
	Ph-GETVALUE confirm	Master and Slave
Event PhL	Ph-EVENT indication	Master and Slave

Temporal Relationships of Service Primitives:

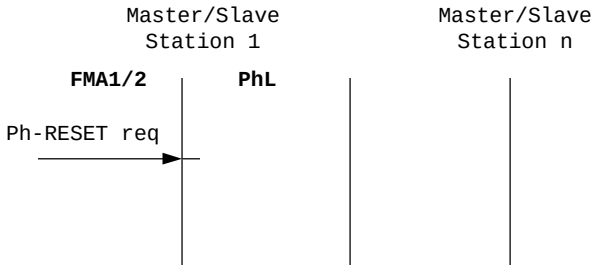


Figure 12: Reset - PhL - Service

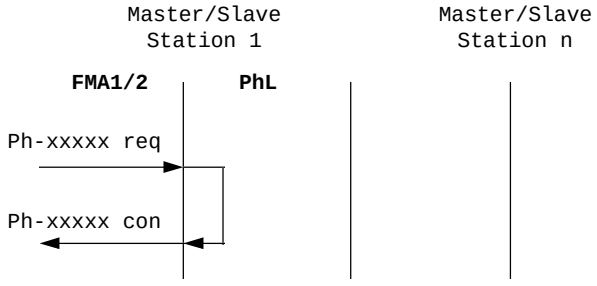


Figure 13: Set Value and Get Value PhL Service (cf Fig. 14 of Part 3)

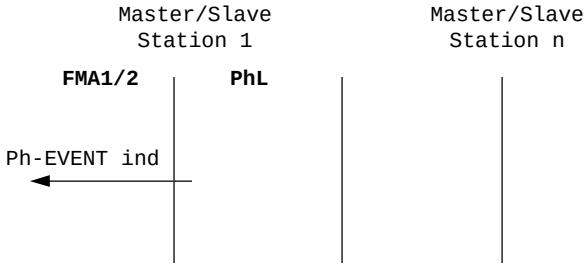


Figure 14: Event PhL Service (cf Fig. 15 of Part 3)

10.3.3 Detailed Specification of Services and Interactions

10.3.3.1 Reset PhL

The service Reset PhL is **mandatory**.

The Ph-RESET request primitive is given to the PhL by the FMA1/2 to reset PhL. The PhL executes this service as described in IEC 1158-2, clause 6. It shall be noted that in contrast to PROFIBUS Management (FMA1/2), no confirmation is returned by PhL (i.e. the FMA1/2 shall generate a confirmation for the FMA1/2-User).

Parameters of the Primitives:

Ph-RESET request

- This primitive has no parameters.

10.3.3.2 Set Value PhL, Get Value PhL

Set Value PhL

The service Set Value PhL is **optional**.

The FMA1/2 passes a Ph-SETVALUE request primitive to a PhL to set a specified variable to a desired value. The PhL will select the variable after receiving the primitives and set it to this value. The FMA1/2 receives a confirmation about this in a Ph-SETVALUE confirm primitive.

Parameters of the Primitives

Ph-SETVALUE request
(parameter name, new value)

- The parameter, parameter name, specifies the variables.

- The parameter, new value, specifies the new value of the variable (cf IEC 1158-2, clause 6):

Table 1: Values of the PhL Variables

Name	Value
Interface mode	- FULL_DUPLEX
	- HALF_DUPLEX
	- DISABLED
Loop-back mode	- in MDS at DTE-DCE interface
	- in MAU near line connection
Preamble extension	- 0 to 7 (preamble extension sequences)
Post-transmission gap extension	- 0 to 7 (gap extension sequences)
Maximum inter-channel signal skew	- 0 to 7 (gap extension sequences)
Transmitter output channel N	- ENABLED
(1 ≤ N ≤ 8)	- DISABLED
Receiver input channel N	- ENABLED
(1 ≤ N ≤ 8)	- DISABLED
Preferred receive channel	- NONE
	- 1 to 8

Ph-SETVALUE confirm
(status)

- The parameter status gives the status of the Ph-SETVALUE request primitive: Success or Failure. The FMA1/2 converts these values into M_status_values "OK" and "NO" for the FMA1/2_SET_VALUE.confirm primitive to the FMA1/2-User.

Get Value PhL

The service Get Value PhL is **optional**.

The FMA1/2 passes a Ph-GETVALUE request primitive to the PhL to read a specified variable. The PhL transfers the value of the variable in a Ph-GETVALUE confirm primitive to the FMA1/2.

Parameters of the Primitives

Ph_GETVALUE.request
(parameter name)

- The parameter, parameter name, specifies the variable.

Ph_GETVALUE.confirm
(current value)

- The parameter, current value, contains the current value that has been requested by the last Ph-GETVALUE request primitive. The values of table 1 shall apply for the variables. In case of a failure, the value "Failure" is given to all variables. FMA1/2 converts this value into M-status-value "NO" for the FMA1/2_Read_Value.confirm primitive to the FMA1/2-User.

10.3.3.3 Event PhL

The service Event PhL is **optional**.

The PhL uses this service to notify the FMA1/2 that variables have been changed their value.

Parameters of the Primitives

Ph-EVENT indication
(parameter name)

- The parameter, parameter name, specifies which variable has changed its value without a request of the FMA1/2 (cf IEC 1158-2, clause 6):

Table 2: Values of the EVENT parameter (parameter name), (cf Table 37 of Part 3)

Parameter name and value
DTE fault
DCE fault

10.4 Coding of the FDL and PhL Variables

The range of values of the FDL and PhL variables are defined in the subclauses 11.2.3.2 and 11.3.3.2 of this part. In addition, a bit coding is given according to the stipulations of the data type in FMA7 (cf subclause 4.8 of Part 7). While using IEC 1158-2 the FDL range of values given in Part 7, subclause 4.5.3 shall be adhered to.

10.4.1 Coding of the FDL-Variables

Baud_rate:

(10) - 31.25 Kbaud

11.4.2 Coding of the Variables

Interface_mode:

(0) - FULL_DUPLEX

(1) - HALF_DUPLEX

Loop_back_mode:

(0) - DISABLED

(1) - in_MDS_at_DTE_DCE_interface

(2) - in_MAU_near_line_connection

Preamble_extension:

(0) to (7) - 0 to 7 octet preamble extension

Post_transmission_gap_extension:

(0) to (7) - 0 to 7 bit times

Maximum_inter_channel_signal_skew:

(0) to (7) - 0 to 7 bit times

Transmitter_output_channel:

MSB LSB

b8	b7	b6	b5	b4	b3	b2	b1
Channel 8	Channel 7	Channel 6	Channel 5	Channel 4	Channel 3	Channel 2	Channel 1

Notation herein for every channel:
(0) - ENABLED
(1) - DISABLED

Figure 15: Encoding of the Transmitter_output_channel

eceiver_input_channel:

MSB LSB

b8	b7	b6	b5	b4	b3	b2	b1
Channel 8	Channel 7	Channel 6	Channel 5	Channel 4	Channel 3	Channel 2	Channel 1

Notation herein for every Channel:
(0) - ENABLED
(1) - DISABLED

Figure 16: Encoding of the Receiver_input_channel

Preferred_receive_channel:
(0) - NONE
(1) - (8) - Channel 1 - Channel 8

10.4.3 List of Object Attributes

In addition to part 7, subclause 4.8, the object attributes defined in table 3 shall be used.

Table 3: Object Attributes

Attribute	Data type
Interface_mode	Unsigned8
Loop_back_mode	Unsigned8
Preamble_extension	Unsigned8
Post_transmission_gap_extension	Unsigned8
Maximum_inter_channel_signal_skew	Unsigned8
Transmitter_output_channel	Unsigned8
Receiver_input_channel	Unsigned8
Preferred_receive_channel	Unsigned8

Appendix A (informative)

Examples of Realizations

A.1 Repeater

The repeater structures described in Part 2, subclause 2-A.1, shall apply furthermore. Instead of the RS-485/TTL receiver and transmitter circuits (Figs. 2-A.1 and 2-A.2 in Part 2), circuits with the technical features (transmitter level and timing, receiver circuit specification) of IEC 1158-2 are planned. Regarding the self-controlled repeater, the directional control should be executed on the one hand at the beginning of the preamble, on the other hand at the end of the end-delimiter or the post-transmission gap.

For a linear bus or tree topology four repeaters may be used at most. It has to be considered thereby that at most 127 stations may be operated with that kind of structure.

A.2 Structures of PROFIBUS Controllers

The structures of PROFIBUS Controllers are designed as described in PROFIBUS Specification "Controllers", with consideration of IEC 1158-2, for connecting a field automation unit (control station, central processing station) or a field device to the transmission medium.

However, it shall be considered that a transceiver and a USRT (Universal Synchronous Receiver/Transmitter) are used according to the technical specifications of IEC 1158-2 instead of a RS-485 transceiver and UART.

A.3 System with several Bus Lines to one Control Station

As described in clause 2-A.3 of Part 2, several bus lines may be used at one control station when applying IEC 1158-2. One USRT device instead of one UART device is needed per bus line (Fig. 2-A.5 of Part 2).

A.4 Redundant Control Station

The structure of a redundant control station as described in clause 2-A.4 of Part 2 is possible.

A.5 Bus Analysis / Diagnostic Unit (Bus Monitor)

The use of this specification makes possible a bus analysis / diagnostic unit as described in clause 2-A.5 of Part 2.

A.6 Intrinsically safe Fieldbus with Power Supply

The intrinsically safe circuits require a limitation of the electrical power, which in turn limits the number of stations.

E.g. for explosive group IIC according to EN 50020, it is assumed that 10 stations may be connected with 10 mA supply current each at $I_K \leq 110$ mA and $U_0 \leq 14$ V.

A.7 Message Rate, System Reaction Time and Token Rotation Time

As described in clause 2-A.6 of Part 2, the message rate R_{SYS} corresponds to the possible number of message cycles per second in the system (cf subclause 4.2.3, equation (23) of Part 4). The maximum system reaction time (also called station or bus access time) for the cyclic send/request (polling) from one master station to n slave stations (master slave system) is computed from the message cycle time and the number of slave stations (cf subclause 4.2.3, equation (24), of Part 4).

The times T are converted into t (seconds) for the following example calculations:

Let t_{SDR}=0.5 ms, t_{ID}=1 ms and DATA_UNIT = 2, 10 and 50 octets with 30 slave stations.

The request frame is in each case without DATA_UNIT (cf subclause 9.6.1, Fig. 8.1, of this part) while the response frame has a variable DATA_UNIT (cf subclause 9.6.3, Fig. 10.2, of this part). Each frame is equipped according to the specifications of IEC 1158-2 with a preamble and a Start Delimiter (1 octet each) and an End Delimiter (1 octet). Furthermore, message repetitions and the times T_{TD} are not considered. Token transfer times are irrelevant, because there is only one master station in the system. The values are rounded.

Table A.1.1: Message Cycle Time

DATA_UNIT	t_{MAC} Time for One Message Cycle [ms] Data Signaling Rate: 31.25 kbit/s
2 Byte	7
10 Byte	9
50 Byte	20

Table A.1.2: Message Rate

DATA_UNIT	R_{SYS} Number of Message Cycles / second [N/s] Data Signaling Rate: 31.25 kbit/s
2 Byte	135
10 Byte	106
50 Byte	51

Table A.1.3: System Reaction Time

DATA_UNIT	t_{SR} System Reaction (Latency) Time with 30 Slave Stations [ms] Data Signaling Rate: 31.25 kbit/s
2 Byte	222
10 Byte	283
50 Byte	590

The same computational basis for the system reaction time is valid for a system with several master stations as described in Part 2, Annex 2-A (equations 2-A.1 and 2-A.2). Figure A.1 shows the bounds of t_{RR} versus the number of master stations and high priority messages at 31.25 kbit/s. As in Part 2, Annex 2-A, message repetitions and the time T_{TD} are not considered. The computations are based upon the token cycle time T_{TC} and the combined token and message cycle time T_{TC} shown in Figs. 2-A.9 and 2-A.10 in Annex 2-A of Part 2.

The following values are assumed for the message cycle:

Send/Request Data with Reply: $T_{S/R}$ = 72 bit (1 octet preamble)

Response with 10 octets DATA_UNIT: $T_{A/R}$ = 176 bit (1 octet preamble)

The following implementation dependent times were chosen:

Token Cycle: at 31.25 kbit/s t_{ID} = 0,5 ms

Token + Message Cycle:

at 31.25 kbit/s t_{ID} = 0,5 ms ; $t_{ID'}$ = 1 ms;

t_{SDR} = 0,5 ms

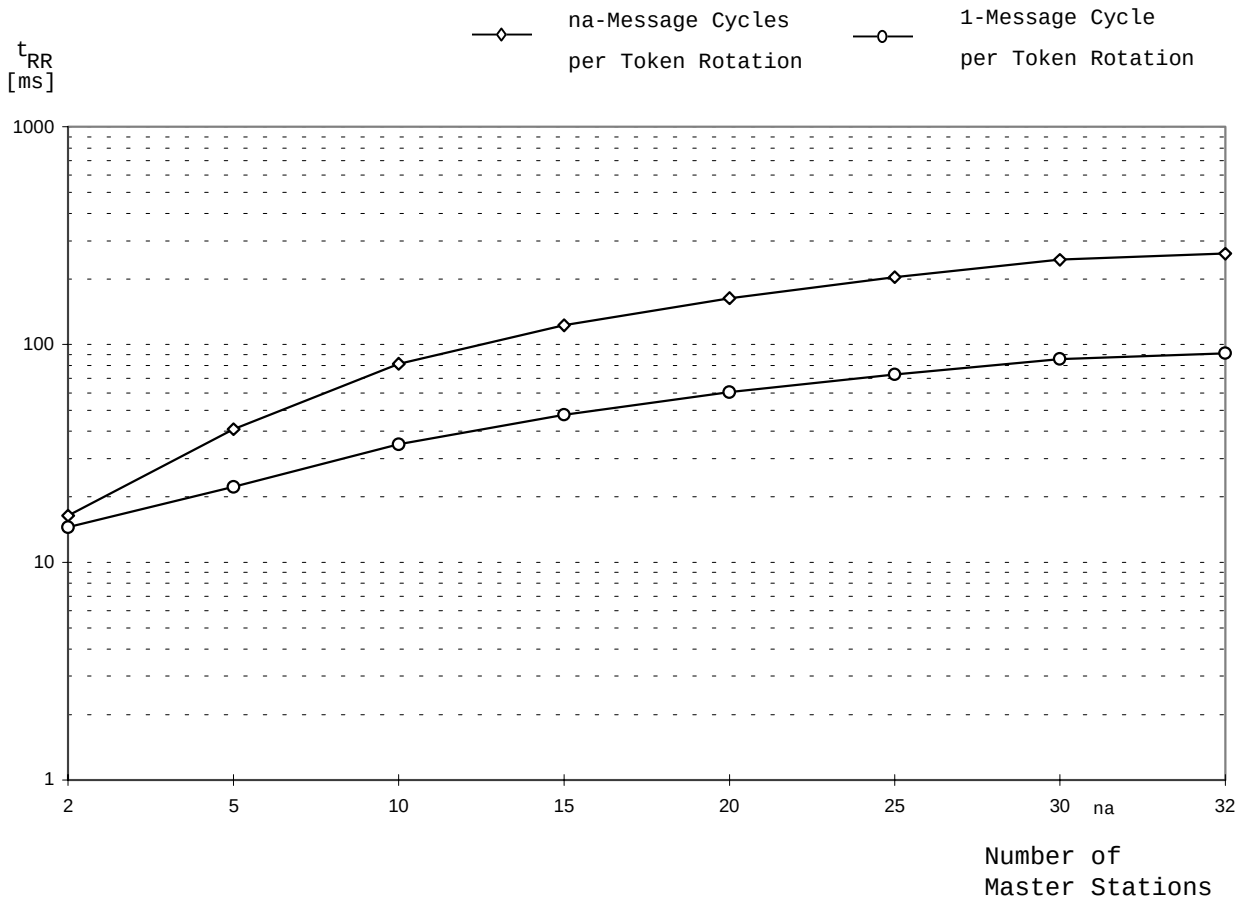


Figure A.1: Real Token Rotation Time t_{RR} (cf Fig. 2-A.13 in Part 2)

PROFIBUS International

Haid- und Neu-Str. 7, 76131 Karlsruhe, Tel.: ++49 721 9658590 Fax: ++49 721 9658589
http://www.profibus.com e-mail: PROFIBUS_International@compuserve.com

638 Members, 5. März 1998, Page 1 (of 2)

Australia (8)

Danfoss
Festo
GEC Alsthom
OSItech
Pepperl + Fuchs
Sensit
SIEMENS
Weidmüller

Austria (23)

ABB
Robert BOSCH
Endress & Hauser
FESTO
GANTNER
ICT/TU Wien
IDEAL
Klöckner Moeller
Landis & Staefa
OMRON
PHÖNIX-Contact
Pilz
SAIA-Burgess
Schiebel
SG Connect
SIEMENS
Technikum Kärnten
Technikum Vorarlberg
TU Wien, Prof. Schildt
Wago
WEIDMÜLLER ConneXt
ZAT
Zimmer

Belgium (17)

Asco Joucomatic
Danfoss
Elbicon
Endress + Hauser
Klöckner Moeller
Katholieke Hogeschool Limburg
Landis & Gyr
Opel Belgium
PEP Modular Computers
Philips/PMA
Promatic-B
Rockwell Automation
SAIA-Burgess
Samson
Schneider
SIEMENS
Technifutur

China (15)

Beijing Materials Handling Research Institute
Beijing Research Institute of Automation for Machinery Industry
Chongquing Institute of Process Automation Instrumentation
Dalian Modular Machine Tool Research Institute
Danfoss Industries
Equipment Design Institute of Dong Feng Motor
Instrumentation Technology and Economy Institute
Shanghai Automation Instrument.
Shanghai Institute of Process Automation Instrumentation
Shanghai OMRON Autom. System
Shenyang Institute of Automation Chinese Academy of Sciences
Siemens Ltd. China
Tianjin Design and Research Institute of Electric Drive
Turck (Tianjin) Sensor
Xian Instrument Faxtory (Grp.) Xiyi

Czech Republic (13)

Automatizace odborny casopis
CTU FEE Praha
Endress - Hauser
ICS s.r.o. (SAIA)
Sidat s.r.o.
Siemens s.r.o. Praha
SWAC Bohemia s.r.o.
Teco a.s.
VAE Controls s.r.o.
WAGO s.r.o.
Weidmueler s.r.o.
ZAT a.s.
ZPA Nova Paka a.s.

Finland (17)

ABB Industry Oy
Advancetec Oy
AEL
Festo
FF-Automation
G&L Beijer Electronics Oy
IFM Electronic Oy
JUHA-Elektro Oy
Klinkmann Automaatio Oy
Kosini Oy Automaatio
Oy Aumator Ab
Oy Danfoss Ab
POHTO
Sensonor Oy
SIEMENS Oy
Vaasa Control Oy
Vaasa Electronics Oy

France (17)

APPLICOM International
Crouzet Automatismes
Ecole d'Ingénieurs de Tours
Endress + Hauser
ENTRELEC
Euro PEP
EUROTHERM Automation
EUROTHERM Vitesse Variable
GIMELEC
Hirschmann
Klöckner-Moeller
Landis & Staefa
MITSUBISHI Electric
SAIA-Burgess
SIEMENS
WAGO
Weidmüller

Germany (185)

ABB Industrietechnik
ABB Kraftwerksleittechnik
AEG Schneider Automation
AGE
Aixo Informationstechnologie
Allen Bradley
AMA Systems
Analog Devices
Apparatebau Hundsbach
Applicom Deutschland
APV Engineering Automation
Artis Ges. f. angew. Meßtechnik
ASP CNC-Technik
AUMA Riester
B & R Industrie Elektronik
Balzers u. Leybold Holding
Barksdale Control
BARTEC
Bauer Antriebstechnik
Baumer Ident
Beckhoff Industrie Elektronik
Bergakademie Freiberg
Bihl + Wiedemann
franz binder
BIZERBA
Robert Bosch
Bürkert
CEAG Sicherheitstechnik
COMSOFT
CONTA-CLIP
Danfoss Antriebs- u. Regeltechnik
Datalogic
Delta t
DEUTA-Werke
Deutschmann Automation
DMS Dorsch Mikrosysteme
DMT Ges. f. Forschung u. Prüfung
DÜCKER Automation
Ebelt
EES-Promotion
ELAU Elektronik Automations
EMG Elektro Mechanik
E.P. Elektro-Projekt
ELTEC Elektronik
Endress + Hauser
Entrelec-Schiele Industriewerke
ERNI Elektroapparate
Esters Elektronik
FachhochschulePforzheim
Fachhochsch. Schmalkalden
FASE
Feller Engineering
Ferrocontrol Steuerungssysteme
FESTO
Ulrich Fock
FZI - Forschungszentrum Informatik
Foxboro Eckardt
FRABA Sensorsysteme
Fraunhofer Gesellschaft IITB
FRIMAT
GE Fanuc Automation
gap - Gesellschaft für angewandte Prozeßdatentechnik
Gesytec
Getriebebau NORD
HARTING
Hartmann und Braun
Dr. Johannes Heidenhain
Hengstler
Hilscher Ges. für Systemautomation
Richard Hirschmann
Hohner Elektrotechnik
Honeywell Regelsysteme
IAM FuE
ibp Elektronik
IEF Werner
ifak e.V. Magdeburg
ifak system
ifm electronic
INDRAMAT
Industrial Research Institute
Inova Computers
INSAT Weitzel & Partner
Institut für Regelungstechnik der RWTH Aachen
Joucomatic
KBR
Kerpenwerk
Kieback & Peter
KIMO Industrie Elektronik
Kisters Maschinenbau
Klöckner Moeller
Klöckner-Holstein-Seitz
Klöpper und Wiege Software
Knick Elektron. Meßgeräte
KONTRON Elektronik
Krohne Meßtechnik
Kuhnke
kws Computersysteme
LAMTEC
Landis&Staefa (Deutschland)
Lang Apparatebau
Lapp Kabel
Lenze, Aerzen
Leuze electronic
Lips
LJU Industrieelektronik
Logic
Loher
Karl Lumberg
Lust Antriebstechnik
Friedrich Lütze
M & R
Matsushita Automation Controls
MESCO Engineering
Messtechnik Sachs
Mettler Toledo
MICRODESIGN
microSYST Microelectronic
Mitsubishi Electric Europe
MTL Instruments
MTS Sensor Technologie
Murrelektronik Automation
Neuberger Gebäudeautomation
Numatics
OMRON Electronics
Adam Opel
Pan Dacom
PAT Pietzsch Automatisierungstechnik
PEP Modular Computers
Pepperl + Fuchs
Philips Industrial Automation Systems
Physikalisch-Techn. Bundesanstalt
Pilz
PMA Prozeß-u. Maschinenautom.
PROMETEC
PROMICON Systems
Reese + Thies GmbH
REO Elektronik
Richardon Consulting
RMP Elektroniksysteme
Rose Elektrotechnik
Rotec Industrieautomation
Rütter & Co.
S.K.I.
SAIA-Burgess Electronics
SAMSON
Sasse Elektronik
Schaeper Automation
Dr. Schenk
Schleicher Relaiswerke
Schneider Electric
Seidel Servo Drives
SEW-EURODRIVE
SIEMENS
SMAR Meß- u. Regeltechnik
SMC Pneumatik
Softing
SPC CIMsoftware
SST Industrial Communication Technology
R. Stahl Schaltgeräte
Max Stegmann
Stein
Steinhoff Automations- und Feldbussysteme
Stöber Antriebstechnik
SÜTRON electronic
SWAC
Technische UNI Dresden
TMG i-tec
TR-Electronic
Trebing & Himstedt Prozeßautomation
Hans Turck
Werner Turck
TWK Elektronik
UNIPO
UNI Dortmund
UNI Erlangen-Nürnberg
UNI Otto-von-Guericke
UNI Wuppertal
VEGA Grieshaber
Verein Deutscher Ingenieure VDI
VIPA
WAGO Kontakttechnik
Weidmüller ConneXt
Wöhrle Industrieelektronik
Wonderware
WZL der RWTH Aachen

Great Britain/UK (35)

Algosystems
APV Automation
Bentley Nevada (UK)
BOC Process Plants
British Federal Ltd.
Burkert Contromatic
Control Technology International
Davis Derby Ltd.
Dwy Cymru Welsh Water
Endress + Hauser
Eurotherm Drives
Hazardous Technical Services
Hirschmann
HSDE
IMI Norgren
ITS Ltd.
Klöckner-Moeller
H. Kuhnke
Kvaerner FSSL Ltd.
Mannesmann Rexroth
Micro Circuit Engineering
Mitsubishi Electric Europe
MTE Ltd.
Nova Weigh Ltd.
Oyster Terminals
PCME
PEP Modular Computers
Practicon
Servomex Plc
St. Helens College
Siemens
Tellima Technology
Wabco Automotice UK

PROFIBUS International

Haid- und Neu-Str. 7, 76131 Karlsruhe, Tel.: ++49 721 9658590 Fax: ++49 721 9658589
http://www.profibus.com e-mail: PROFIBUS_International@compuserve.com

638 Members, 5. März 1998, Page 2 (of 2)

WAGO
Weidmuller Connext

Italy (37)

A.S.CO Automazione
ASCO Joucomatic
AUMA Italiana
BIT
Camozzi
Carpaneto
Crouzet Componenti
Danfoss
Eco Automazione
Endress+Hauser Italia
ESA Elettronica
EXOR
Festo
Fitre
GEFRAN Sensori
Hirschmann-Alhof
I.M.A. (Italia Manutenzione Automatismi)
Klökner Moeller
Landis & Staefa
MATRIX
Matsushita Automation Controls
Metal Work
MODULTRONIC
OMRON Electronics
Orsi Automazione
Pepperl + Fuchs
Proseat
Rotomec Automation
SAIA-Burgess
SIEI
Siemens
SMC Italia
Telmotor
Tesy
TEX Computer
Treesse Progetti
Weidmüller

Japan (15)

Churitsu Electric Corp.
CKD Corporation
C. Correns & Co.
Fuji Electric Co.
Hitachi Ltd.
Nihon Weidmueller Co.
Oriental Motor Co.
Sakura Endress Co.
Siemens K. K.
SMC Corporation
Sumitomo Heavy Industry
SUNX Ltd.
Toshiba Corp. Fuchu Works
Toyoda Machinery
Yaskawa Electric

Netherlands (26)

ABB Componenten
B & R Industriele Automatisering
Endress+Hauser
Festo
Geveke Electronics
Hirschmann
ICONICS Europe
Industrial Automation Link
ITHO
Klöckner-Moeller
Landis & Staefa (Nederland)
Leuze Electronic
PEP Modular Computers
Philips
Phoenix Contact
Praxis Automation Tech.
Rockwell Automation
SAIA Burgess
Samson Regeltechniek
Siemens Nederland
THIS Automation
Turck
Visolux Holland
Wago Regoort
Weidmüller
Wizcon Nederland

Norway (35)

4tech AS
ABB Teknoloki AS
AD Elektronikk AS
Autic Systems
Automasjon og Styringsystemer
Beijer Electronics AS
Dyno Industrier
Elva Induksjon
Endress + Hauser
Fellesmeieriet Oslo
Festo AB
Hitec ASA
Hydro Aluminium
IGP
IMI Norgren AS
Klöckner-Moeller Norsk AS
J. F. Knudtzen AS
Kongsberg Simrad AS
Konsulent K. Undbekken AS
Kystdirektoratet
Linkcom AS
Malthe Winje Automasjon AS
Marintek AS
Micro-Control AS
MurrElektronikk AS
NSI Follum Fabrikker
Odim Skodje AS
Rexroth Mecman
SCA Mölnlycke
Scan-Sense AS
Siemens AS
System Engineering
Triple-S Industry
Ulstein Automation AS
Walbro Automotive

Russia/GUS (5)

Central Power Energy
GOSNIIAS
RTSoft
SIEMENS
Uralenergo

Sweden (37)

Åkerström Björbo AB
Alfa Laval Automation AB
AMBITRON AB
Autic System AB
Beijer Electronics AB
BTG Källe Inventing AB
Bûrkert-Contromatic AB
Danfoss AB
Elektro Logik AB
Endress+Hauser Stockholm
ERDE-Elektronik AB
ERICHS ARMATUR AB
FESTO AB
GERMATECH AB
Hellermann Scandinavia AB
HMS Fieldbus Systems AB
Högskolan i Kalmar
Jor AB
Kuhnke Automation AB
Leine & Linde AB
Malthe Winje Automation AB
MILTRONIC AB
Nobel Elektronik AB
Parker Hannifin AB
P&L Automatik AB
PEP Modular Computers AB
Pepperl + Fuchs AB
Rexroth Mecman AB
Rudbecksgymnasiet
Sensor Control Nordic AB
SDT AB
SIEMENS AB
SMC Pneumatics Sweden AB
Stohne Elteknik AB
Tillquist Process AB
TR Electronic Sweden AB
Weidmuller Eurolon

Switzerland (32)

ABB Normelec AG
Autronic AG
B + R Industrie-Automation
Robert Bosch AG
Bünger Consulting
Bürkert Contromatic
CIM Center Aargau
Digitrade AG
ECONOTEC
Endress + Hauser
Erni Elektrotechnik
FESTO
Hartmann + Braun SA
Indumo Software Eng.
Ingenieurschule Bern HTL
IST Engineering
Klöckner-Moeller
Krohne AG
Landis & Gyr
Müller Systemtechnik
Murrelektronik
OMRON Electronics
Prola AG
Rockwell Automation
SAIA Burgess Electronics
Siemens Schweiz
SMC Pneumatic
Static Input System
SWAN
WAGO Contact
Weidmüller C. Geisser
Züllig

South Africa (19)

Actum
Adroit Technologies
AUMA
Burkert Contromatic
Danfoss
Directech
Endress + Hauser
Ernest Lowe
Futuristix
Groupe Schneider
Hyflo
IMS Projects
MDA Instruments
Measuretronic
Process Automation
Shorrock Senior Ass.
Siemens Ltd.
Transvaal Sugar
Xycom

USA (103)

ABB Flexible Automation
AcuTek Automation
Applicom International
Arc Informatique
ASCO Pneumatic Controls
AVG-UTICOR
AWC Incorporated
Axiomatic Technologies
Baldor Electric Company
Balluff
Belden Wire&Cable
Burkert Contromatic Corp.
Cleveland Motion Controls
Compressor Controls
Control Corp. of America
Controlware Corp.
Cummins Engine Co.
D.I.P. Inc.
Danfoss Electronic Drives
Delta Computer Systems
Dorner Mfg. Corp.
EATON Corporation
Electric Depot
EMS
Endress + Hauser Instruments
ERNI Components Inc.
Escort Memory Systems
EXOR Electronic R&D
Fairbanks Scales
Festo Corporation
GE Fanuc Automation
General Motors Power Train
Great Lakes Controls
Hardy Instruments
Hilscher Automation
Hirschmann Rheinmetall
HMS Fieldbus Systems
Horner Electric
ICP Panel-Tec
ICT Inc.
ifak
IFS Industrial Communications
Industrial Systems Design
Intellution
Interlink
Keystone Controls
Klockner-Moeller
Logical Design Group
Lumberg Inc.
Lutze Inc.
MagneTek Corp.
Magnetrol International
Mannesmann Rexroth
Mettler-Toledo
MicroSmith
Milltronics Ltd.
Mitsubishi Electric Automation
Moore Products Co.
MTS Systems Corp.
Murrelektronik Inc.
Nemasoft
Nematron Corp.
New Era Controls
Numatics Inc.
OLFlex Wire & Cable
Orchid Technologies
P & L Automatik
Parijat Controlware
Parker Hannifin
PEP Modular Computers
Pepperl + Fuchs Inc.
Philip Morris USA
Power Measurement
Professional Control
ProSoft Technology
RDE Connectors & Cables
Rexroth Indramat
RTC Group
Samson Controls
SHIP STAR Associates
Siemens E & A
SISCO Inc.
Smar Research Corp.
Softing GmbH
SoftPLC Corp.
Spectrum Controls
SST, Inc.
Steeplechase Software
Synergetic MicroSystems
Sytek
Taylor Industrial Software
Think & Do Software
Total Control Products
TR Encoder Solutions
TRS Fieldbus Systems
Viking Electric
VITA
WAGO Corporation
Weidmuller Inc.
Wieland Electric
Wonderware Corp.
Daniel Woodhead
Yaskawa Electric America

BookShop
PROFIBUS Nutzerorganisation e.V.
valid from 01.04.1998

Title	Prod-No	Language	Price per copy (DM)	
PROFIBUS Standard			**Non-members**	**Members**
PROFIBUS Specification (FMS, DP, PA) All normative parts of the PROFIBUS Specification. According to the European Standard EN 50 170 vol. 2, app. 1000 pages	**0.032**	**English**	**500,-**	**250,-**
PROFIBUS-Guidelines				
Implementation Guide to DIN 19245 Part 1 Additional specifications approx. 20 pages	**2.001**	**German**	**50,-**	**20,-**
Implementation Guide DIN 19 245 Part 2 Additional specifications approx. 40 pages	**2.011**	**German**	**100,-**	**50,-**
Implementation Guide DIN E 19 245 Part 3 Additional specifications incl. new GSD-Formats, approx. 70 pages	**2.041**	**German**	**100,-**	**50,-**
Test Specifications for PROFIBUS DP-Slaves Specification of test procedure for Certification of DP-Slaves, app. 30 p.	**2.032**	**English**	**100,-**	**50,-**
Test specifications for PROFIBUS-DP Masters Specification of test procedure for Certification of DP-Masters,app. 30 p.	**2.071**	**German**	**100,-**	**50,-**
Fibre optical data transfer for PROFIBUS Specification, media characteristics, connectors, approx. 60 pages	**2.022**	**English**	**100,-**	**50,-**
***New:* PROFIBUS-DP Extensions** DP Extensions to EN 50170, approx. 330 pages	**2.082**	**English**	**500,-**	**250,-**
PROFIBUS-PA User and Installation Guideline Technical guidance for the use of IEC 1158-2 with PROFIBUS-PA	**2.091**	**German**	**400,-**	**200,-**
***New:* GSD Extensions for PROFIBUS-FMS** Definition of the GSD-File formats for FMS, app. 100 pages	**2.101**	**German**	**200,-**	**100,-**
***New:* Installation Guideline for PROFIBUS-FMS/DP** Installation and wiring reccommendations for RS 485 Transmission app. 30 pages	**2.111** **2.112**	**German** **English**	**60,-**	**30,-**
PROFIBUS Profiles				
Profile for Communication between Controllers FMS-Communication profile, specification of required services,app.20 p.	**3.002**	**English**	**50,-**	**20,-**
Profile Building Automation FMS-Branch profile, specification of device classes, app. 300 pages	**3.011**	**German**	**200,-**	**100,-**
Profile for Sensors and Actuators FMS - Communication profile and device data sheets, app. 100 pages	**3.021**	**German**	**150,-**	**70,-**
***New:* Profile for Process Automation Class A+B** Branch profile for Process Automation devices, app.100 pages	**3.042**	**English**	**500,-**	**250,-**
Profile for NC/RC Controllers DP profile for NC/RC Controllers, app. 20 pages	**3.051** **3.052**	**German** **English**	**100,-** **100,-**	**50,-** **50,-**
Profile for Encoders DP profile for rotary, angle and linear encoders, app. 50 pages	**3.062**	**English**	**100,-**	**50,-**
***New:* Profile for Variable Speed Drives** FMS-/DP-Profile for electric drive technique, app. 90 pages	**3.071**	**German**	**150,-**	**70,-**
***NEU:* Profile for HMI Systems** DP-Profile for Human Machine Interface systems, app. 50 pages	**3.081**	**German**	**100,-**	**50,-**
Brochures and Catalogues				
Technical brochure PROFIBUS short description PROFIBUS-FMS and -DP and -PA, 32 pages	**4.001** **4.002**	**German** **English**	**350,-**/100pcs* **350,-**/100pcs*	**175,-**/100pcs* **175,-**/100pcs*
***New:* PROFIBUS for Process Automation** Image Brochure PROFIBUS-PA , 6 pages, 4 colours	**4.031** **4.032**	**German** **English**	**240,-**/100pcs* **240,-**/100pcs*	**120,-**/100pcs* **120,-**/100pcs*
***New:* PROFIBUS Products and Services** The Electronic Product Guide on CD ROM for MS-Windows©	**4.090**	**German &** **English**	**350,-**/100pcs*	**350,-**/100pcs*
Technical Literature for Workshops, Training etc.				
PROFIBUS Public 5 Technical Press Reports from Germany, USA, Europa, Asia, app. 280 p.	**4.041**	**German &** **English**	**60,-**	**30,-**
Slide-Set PROFIBUS-FMS, -DP, -PA 37 4-colour slides with technical details, description + PowerPoint File	**4.051** **4.052**	**German** **English**	**500,-** **500,-**	**275,-** **275,-**
R. Busse, Feldbussysteme im Vergleich Specialized book, Comparison of fieldbus-systems: PROFIBUS, Interbus, CAN ...	**4.061**	**German**	**44,-** fix price incl. VAT	**44,-** fix price incl VAT
M. Popp, The Rapid Way to PROFIBUS-DP Specialized book, for developers of field devices and plant owners	**4.071** **4.072**	**German** **English**	**48,-*** **48,-***	**30,-*** **30,-***
***New:* Slide Set PROFIBUS-PA Applications** CD with 26 4c colour slides of installations at Wacker,Shell, Bitburger (CorelDraw 5)	**4.082**	**English**	**150,-**	**75,-**

*other amounts on request!

order form see reverse side...

PROFIBUS Nutzerorganisation e.V. Office: Haid-und-Neu-Straße 7, D-76131 Karlsruhe, Tel.: ++721 / 9658-590, Fax: ++721 / 9658-589,
Bank Details: Baden-Württembergische Bank AG Karlsruhe, Bank Code: 660 200 20, Account Number: 403 57667 00

Order-form, by fax

PROFIBUS Nutzerorganisation e. V., Haid-und-Neu-Straße 7, D-76131 Karlsruhe
Fax: ++(0)721-9658-640
Phone: ++(0)721-9658-590

please supply the following items:

	Title	Prod.-No.	Language	Qty	Price per copy	Total
1.						
2.						
3.						
4.						
5.						

Our VAT identification no.:_____

Delivery address:	**Invoice address:**
Company	Company
Department	Department
Name	Name
Phone / Fax-No.	Phone / Fax - No.
Street/P.O. Box	Street/P.O. Box
Country / Town / Postcode	Country / Town / Postcode
Company stamp:	Date / Signature

Visit the PROFIBUS Web Site under: http://www.profibus.com

Terms of payment and delivery:

Orders and deliveries in the Federal Republic of Germany: All prices are net prices, excluding packaging, shipping and VAT. Payn net 14 days - no discount.The invoice is your order confirmation. The items are property of PROFIBUS Nutzerorganisation e.V. until fu payment is settled. The minimum order value for non-members is 80,-- DM

Orders and deliveries to foreign countries: In addition to the above mentioned terms of payment the following terms are in force. Members' Prices are only granted to all paid-up members of PROFIBUS Nutzerorganisation Germany. Members of other Regional PROFIBUS User Organisations are treated as non-members. The Regional PROFIBUS User Organisations will grant the usual discou to their members. Shipments to foreign countries are always dispatched by United Parcel Service - we will charge you the full UPS shi costs as well as bank fees. The minimum order value for orders from abroad is DM 150,--. Please note that orders from EC countries without VAT - Identification Number can not be processed.

PROFIBUS INTERNATIONAL Support

Technical Support Center
Haid-und-Neu-Str. 7, D-76131 Karlsruhe
Phone: ++49 721 9658590 Fax: ++49 721 9658589
e-mail: PROFIBUS_International@compuserve.com
http://www.profibus.com

Regional PROFIBUS User Organizations

Australien / Australia
Australian PROFIBUS User Group
c/o OSItech Pty. Ltd.
Mr. Michael Gough
P.O. Box 315
AUS-Kilsyth, Vic. 3137
Phone: ++61 3 9761 5599
Fax: ++61 3 9761 5525
profibus@fieldbus.com.au

Brasilien / Brazil (in foundation)
Association PROFIBUS Brazil
c/o Siemens Ltda PSIAB
Mr. Eng. Juan Rodriguez
R. Cel. Bento Bicudo, 111
BR-05069-900 Sao Paolo, SP
Phone: ++11 833 49 35
Fax: ++11 833 4772
juanr@siemens.br.scn.de

Belgien / Belgium
PROFIBUS Belgium
Mr. Herman Looghe
Lakenweversstraat 21
B-1050 Brussels
Phone: ++32 2 510 2521
Fax: ++32 2 510 2561
herman.looghe@fabrimetal.be

China
Chinese PROFIBUS User
Organisation, c/o China Ass. for
Mechatronics Technology and
Applications, Mrs. WangJun
1 Jiaochangkou Street
Deshengmenwai
PRC-100011 Bejing, CHINA
Tel.: 0086 10 62 02 92 18
Fax: 0086 10 62 01 78 73
profibus@public.bta.net.cn

Czech Republic
PROFIBUS Association Czech
Republic
(in foundation)
Mr. Zdenek Hanzalek
Karlovo nam. 13
CZ-12135 Prague 2
Tel.: 00420 2 2435 74 34
Fax: 00420 2 2435 72 98

Deutschland / Germany
PROFIBUS Nutzerorganisation e.V.
Mr. Michael Volz
Haid-und-Neu-Str. 7
D-76131 Karlsruhe
Phone: ++49 721 96 58 590
Fax: ++49 721 96 58 589
PROFIBUS_International@compuserve.com

Finnland / Finland
PROFIBUS Finland
c/o AEL Automaatio
Mr. Taisto Kaijanen
Kaarnatie 4
FIN-00410 Helsinki
Phone: ++35 8 9 5307259
Fax: ++35 8 9 5307360
taisto.kaijanen@ael.fi

Frankreich / France
France PROFIBUS
Mrs. Christiane Bigot
4, rue des Colonels Renard
F-75017 Paris
Phone: ++33 1 45 74 63 22
Fax: ++33 1 45 74 03 33
france.profibus@wanadoo.fr

Großbritannien / United Kingdom
The PROFIBUS Group
Mr. Bob Squirrell
1, West Street
GB-P014 4DH Titchfield, Hants
Phone: ++44 1329 843043
Fax: ++44 1329 512063
bobsq@dial.pipex.com

Italien / Italy
PROFIBUS Network Italia
Ms. Analisa Pesarin
Corso Spagna 12
I-35127 Padova
Phone: ++39 49 806 1214
Fax: ++39 49 870 3255
pni@iperv.it

Japan / Japan
Japanese PROFIBUS Organisation
Mr. Hideki Nakamichi
Takanawa Park Tower 18F
3-20-14 Higashi-Gotanda
J-#141 Shinagawa-ku, Tokyo
Tel.: ++81 / 3 / 54 23 85 90
Fax: ++81 / 3 / 54 23 8734
masahiko.imoto@skk.tyo1.siemens.net

Niederlande / Netherlands
PROFIBUS Nederland
p/a Vereniging FME
Herr Aat C. van der Giessen
Boerhaavelaan 40
NL-2713 HX Zoetermeer
Phone: ++3179 3 53 13 53
Fax: ++3179 3 53 13 65
profibus@fme.nl

Norwegen / Norway
PROFIBUS User Organisation Norway
c/o AD Elektronik AS
Mr. Kai-Atle Myrvang
Lunahuset Berghagan, P.O. Box 104
N-1405 Langhus
Tel.: ++47 64 86 9970
Fax: ++47 64 86 9920
profibus@ade.no

Österreich / Austria
PROFIBUS Nutzerorganisation Österreich
c/o TU Wien, Institut für Computertechnik
Prof. Dietrich
Gusshausstr. 25 - 29
A-1040 Wien
Phone: ++43 1 588013829
Fax: ++43 1 505389814
pno@ict.tuwien.ac.at

Rußland / Russia
PROFIBUS User Organisation Russia
c/o Vera + Association
Mrs. Olga Sinenko
P.O. Box 159
Pervomaiskaya str, 109/2
105203 Moscow, Russia
Phone: ++7 095 465-55-53
Fax: ++7 095 742-68-29
pno@veraplus.msk.ru

Schweden / Sweden
PROFIBUS i Sverige
c/o P & L Automatik AB
Mr. Peter Bengtsson
Kommandörsgatan 3
S-28135 Hässleholm
Phone: ++46 4 51 49 460
Fax: ++46 4 51 89 833
kansli@pis.se

Schweiz / Switzerland
PROFIBUS Nutzerorganisation
Schweiz
Ms. Karin Beyeler
Fritz-Käserstr. 10
CH-4562 Biberist
Phone: ++41 32 672 49 15
Fax: ++41 32 672 49 17
profibus@thenet.ch

Singapore/Malaysia/Thailand
PROFIBUS Singapore
(in foundation)
c/o Weidmüller Pte Ltd.
Herr Dipl.-Ing. Erich Vosshage
1, Kallang Sector #07-01/03
SGP-349276 Singapore
Tel.: 0065 - 841 53 11
Fax: 0065 - 841 5377

Südafrika / South Africa
PROFIBUS User Organisation
SouthAfrica
c/o Endress & Hauser
Mr. John Immelmann
Postfach 783996
RSA-2146 Sandton
Phone: ++27 11 444 1386
Fax: ++27 11 444 4032
endress@icon.co.za

USA
PROFIBUS Trade Organization, PTO
Mr. Michael Bryant
5010 East Shea Blvd., Suite C-226
Scottsdale, AZ 85254-4683 USA
Phone: ++1 602 483 2456
Fax: ++1 602 483 7202
mbryant@netzone.com

PROFIBUS Nutzerorganisation e.V.
Haid-und-Neu-Str. 7
D-76131 Karlsruhe
Phone: ++ 721 / 96 58 590
Fax: ++ 721 / 96 58 589

http://www.profibus.com
e-mail: PROFIBUS_International@compuserve.com