ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Kontron / PEP VM30* at our website: *Click HERE*



PROFIBUS Protocol Software Layer 2

for MC68302-based Controllers

OS-9/68K

Version 3.12

User's Manual
Issue 2

 © 1995 PEP Modular Computers

REVISION HISTORY
PROFIBUS Protocol Software Layer 2 User's Manual
Version 3.12

Issue	Brief Description of Changes	S/W Index	Date of Issue
1	First Issue	3.1	June, 1993
1.0.1	Corrections to Chapter 4	3.1	April, 1994
2	Updated to Version 3.12	3.12	February, 1995

This document contains proprietary information of **Softing GmbH**, translated and reproduced under license by **PEP *Modular Computers***. It may not be copied or transmitted by any means, passed to others, or stored in any retrieval system or media, without the prior consent of **PEP *Modular Computers*** or its authorized agents.

The information in this document is, to the best of our knowledge, entirely correct. However, **PEP *Modular Computers*** cannot accept liability for any inaccuracies, or the consequences thereof, nor for any liability arising from the use or application of any circuit, product, or example shown in this document. **PEP *Modular Computers*** rely on the originator of the Software for information contained in this manual and consequently cannot ensure that the information is correct or contains changes which **PEP *Modular Computers*** have not been informed of.

PEP *Modular Computers* reserve the right to change, modify, or improve this document or the product described herein, as seen fit by **PEP *Modular Computers*** without further notice.

PEP *Modular Computers*® Two Year Limited Warranty

We grant the original purchaser of **PEP** products the following hardware warranty. No other warranties that may be granted or implied by anyone on behalf of **PEP** are valid unless the consumer has the express written consent of **PEP *Modular Computers***.

PEP *Modular Computers* warrants their own products (excluding software) to be free from defects in workmanship and materials for a period of 12 consecutive months from the date of purchase. This warranty is not transferable nor extendible to cover any other consumers or long term storage of the product.

This warranty does not cover products which have been modified, altered, or repaired by any other party than **PEP *Modular Computers*** or their authorized agents. Furthermore, any product which has been, or is suspected of being damaged as a result of negligence, misuse, incorrect handling, servicing or maintenance; or has been damaged as a result of excessive current/voltage or temperature; or has had its serial number(s), any other markings, or parts thereof altered, defaced, or removed will also be excluded from this warranty.

A customer who has not excluded his eligibility for this warranty may, in the event of any claim, return the product at the earliest possible convenience, together with a copy of the original proof of purchase, a full description of the application it is used on, and a description of the defect; to the original place of purchase. Pack the product in such a way as to ensure safe transportation (we recommend the original packing materials), whereby **PEP** undertakes to repair or replace any part, assembly or sub-assembly at our discretion; or, to refund the original cost of purchase, if appropriate.

In the event of repair, refund, or replacement of any part, the ownership of the removed or replaced parts reverts to **PEP *Modular Computers*,** and the remaining part of the original guarantee, or any new guarantee to cover the repaired or replaced items, will be transferred to cover the new or repaired items. Any extensions to the original guarantee are considered gestures of goodwill, and will be defined in the "Repair Report" returned from **PEP** with the repaired or replaced item.

Other than the repair, replacement, or refund specified above, **PEP *Modular Computers*** will not accept any liability for any further claims which result directly or indirectly from any warranty claim. We specifically exclude any claim for damage to any system or process in which the product was employed, or any loss incurred as a result of the product not functioning at any given time. The extent of **PEP *Modular Computers*** liability to the customer shall not be greater than the original purchase price of the item for which any claim exists.

PEP *Modular Computers* makes no warranty or representation, either express or implied, with respect to its products, reliability, fitness, quality, marketability or ability to fulfill any particular application or purpose. As a result, the products are sold "as is," and the responsibility to ensure their suitability for any given task remains the purchaser's.

In no event will **PEP** be liable for direct, indirect, or consequential damages resulting from the use of our hardware or software products, or documentation; even if we were advised of the possibility of such claims prior to the purchase of, or during any period since the purchase of the product.

Please remember that no **PEP *Modular Computers*** employee, dealer, or agent are authorized to make any modification or addition to the above terms, either verbally or in any other form written or electronically transmitted, without consent.

 © 1995 PEP Modular Computers

TABLE OF CONTENTS

Page

1. Introduction 1-1
1.1 Scope 1
1.2 Documentation References 1
1.3 Ordering Information 2

2. Function and Architecture 2-1
2.1 Basic Properties 1
2.2 Protocol Architecture 1
Figure 2.2.0.1: PROFIBUS Protocol Architecture 2
2.3 Layer 1 (Physical Layer) 3
Table 2.3.0.1: RS-485 Transmission Technique 3
2.4 Layer 2 (Data Link Layer) 4
2.4.1 Overview 4
Table 2.4.1.1 Data Transmission Services of Layer 2 5
Table 2.4.1.2 Technical Features of Layers 1 and 2 6
2.4.2 Implementation Overview 6
2.4.3 Resource Circulation 7

3. Hardware Configuration 3-1
3.1 PROFIBUS Controllers 1
Table 3.1.0.1: CPU and Controller Characteristics 1
3.2 The MC68302 IMP (Integrated Multiprotocol Processor) 2
3.2.1 Overview 2
3.2.2 Microprogramming 2
3.2.3 Using Internal Function Groups of the MC 68302 2
3.2.4 External Wiring of the MC68302 3
3.3 PROFIBUS Physical Layer 4
3.3.1 Version 1 4
Table 3.3.1.1 Electrical Characteristics 4
Figure 3.3.1.2 Repeater in Linear Bus Topology 6
Table 3.3.1.3 Connector Pin Assignments and Layout 7
3.4 SC-485F Serial Communications Controller (SCC) Configuration 8

4. Software Architecture 4-1
4.1 OS-9 File System and Architecture 1
Figure 4.1.0.1: OS-9 Software Architecture 6
4.2 PROFIBUS Installation 7
4.2.1 Starting PROFIBUS 7
4.2.2 Running PROFIBUS on a VM30 System 8
4.2.3 Running PROFIBUS on a VIUC System 9
4.2.4 Testing the PROFIBUS Connection 10
4.3 Intercommunication Interface 11
4.4 PROFIBUS Library "pbl2hlf.l" 12
4.4.1 General Functions 13
open_PROFI 13
close_PROFI 13
4.4.2 Data Transfer Functions 14
open_JOB (Obsolete) 15
open_JOB_S 17
open_JOB_R_SDX 19
open_JOB_R_SRD 21
close_JOB 23

 © 1995 PEP Modular Computers

send_SDA . . . 24
send_SDN . . . 25
send_SRD . . . 26
send_RPLUPD_S . . . 28
send_RPLUPD_M . . . 29
ready_IND . . . 30
receive_IND . . . 30
release_IND . . . 32
4.4.3 Management Functions . . . 33
get_LAS . . . 33
get_CTR . . . 34
get_TRR . . . 35
enable_EVENT . . . 36
disable_EVENT . . . 37
4.5 FDL Interface Library "pbl2llf.l" . . . 39
4.5.1 Function fdl_open . . . 39
4.5.2 Function fdl_req . . . 39
4.5.3 Function fdl_con_ind . . . 39
4.5.4 Function fdl_con_ind_poll . . . 40
4.5.5 Function fdl_close . . . 40
4.5.6 FDL Services . . . 40
Figure 4.5.6.1: SDA Service . . . *42*
Figure 4.5.6.2: SDN Service . . . *42*
Figure 4.5.6.3: SRD Service . . . *43*
Figure 4.5.6.4 Start of CSRD Service . . . *44*
Figure 4.5.6.5 End of CSRD Service . . . *45*
SDA (Send Data with Acknowledge) Request . . . 46
SDA (Send Data with Acknowledge) Confirmation . . . 48
SDA (Send Data with Acknowledge) Indication . . . 50
SDN (Send Data with No Acknowledge) Request . . . 52
SDN (Send Data with No Acknowledge) Confirmation . . . 54
SDN (Send Data with No Acknowledge) Indication . . . 56
SRD (Send and Request Data with Reply) Request . . . 57
SRD (Send and Request Data with Reply) Confirmation . . . 59
SRD (Send and Request Data with Reply) Indication . . . 61
REPLY_UPDATE Request . . . 63
REPLY_UPDATE Confirmation . . . 65
SEND_UPDATE Request . . . 67
SEND_UPDATE Confirmation . . . 69
LOAD_POLL_LIST Request . . . 71
LOAD_POLL_LIST Confirmation . . . 73
Cyclic Send and Request Data with Reply (CSRD) Confirmation . . . 74
POLL_ENTRY Request . . . 76
POLL_ENTRY Confirmation . . . 77
DEACT_POLL_LIST Request . . . 78
DEACT_POLL_LIST Confirmation . . . 79
4.5.7 FMA Services . . . 80
FMA2_RESET Request . . . 82
FMA2_RESET Confirmation . . . 83
FMA2_SET_BUSPARAMETER Request . . . 84
FMA2_SET_BUSPARAMETER Confirmation . . . 86
FMA2_CHANGE_BUSPARAMETER Request . . . 87
FMA2_CHANGE_BUSPARAMETER Confirmation . . . 89
FMA2_SET_STATISTIC_CTR Request . . . 90
FMA2_SET_STATISTIC_CTR Confirmation . . . 91
FMA2_READ_BUSPARAMETER Request . . . 92

 © 1995 PEP Modular Computers

FMA2_READ_BUSPARAMETER Confirmation . . . 93
FMA2_READ_STATISTIC_CTR Request . . . 95
FMA2_READ_STATISTIC_CTR Confirmation . . . 96
FMA2_READ_TRR Request . . . 97
FMA2_READ_TRR Confirmation . . . 98
FMA2_READ_LAS Request . . . 99
FMA2_READ_LAS Confirmation . . . 100
FMA2_READ_GAPLIST Request . . . 101
FMA2_READ_GAPLIST Confirmation . . . 102
FMA2_EVENT Indication . . . 103
FMA2_IDENT Request . . . 104
FMA2_IDENT Confirmation . . . 106
FMA2_LSAP_STATUS Request . . . 108
FMA2_LSAP_STATUS Confirmation . . . 110
FMA2_LIVELIST Request . . . 112
FMA2_LIVELIST Confirmation . . . 113
FMA2_ACTIVATE_SAP Request . . . 114
FMA2_ACTIVATE_SAP Confirmation . . . 116
FMA2_ACTIVATE_RSAP Request . . . 117
FMA2_ACTIVATE_RSAP Confirmation . . . 119
FMA2_DEACTIVATE_SAP Request . . . 120
FMA2_DEACTIVATE_SAP Confirmation . . . 121
4.5.8 Services for the Administration of the Resources . . . 123
WAIT_FOR_FMA2_EVENT Request . . . 124
WAIT_FOR_FMA2_EVENT Confirmation . . . 125
WITHDRAW_EVENT Request . . . 126
WITHDRAW_EVENT Confirmation . . . 127
PUT_RESRC_TO_FDL Request . . . 128
PUT_RESRC_TO_FDL Confirmation . . . 130
WITHDRAW_RESRC_FROM_FDL Request . . . 131
WITHDRAW_RESRC_FROM_FDL Confirmation . . . 132
4.5.9 Parameterizing Layer 2 . . . 135

5. Release Notes . . . 5-1

Appendix A Status Values . . . A-1

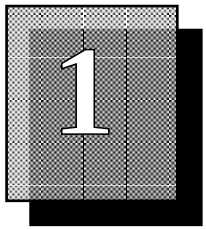
Appendix B Definition of Constants . . . B-1

Appendix C. Type Definitions . . . C-1

Appendix D. Demo Examples . . . D-1

Appendix SCC. Serial Communications Controller

 © 1995 PEP Modular Computers



1. INTRODUCTION

This manual describes the implementation of the PROFIBUS layer 2 protocol software running under the realtime kernel/operating system OS-9 and PEP's MC68302 controllers (i.e. IUC/ VIUC/ VM30/ SMART I/O).

OS-9 extensions allow programming in the usual way. The layer 2 library allows the user to use PROFIBUS services without complex programming sequences, reducing the time to get the layer 2 application to a maximum.

The direct connection to the OS-9/NFM (Network File Manager) supports features such as transparent file access, loading of tasks, remote login and remote source level debugging. With these features, application and communication tasks running on an intelligent I/O node can be tested and debugged from a host computer (e.g. a VME system).

The topics described in this manual include:

- *Functional description of the software architecture*
- *Guidance for installation, hardware adjustment and start up of the communications software*
- *Descriptions of the PROFIBUS objects and services*
- *Description of the communication interface and the layer 2 libraries*
- *OS-9 implementation*
- *Application example*

1.1 Scope

This implementation is based on the PROFIBUS Standard DIN 19245, Part 1 from April 1991.

The implementation encompasses:

- *all communication services,*
- *all (including the options) management services,*
- *multi-master functionality (for up to 127 participants),*
- *full address expansion (64 Service Access Points, 64 segment addresses).*

1.2 Documentation References

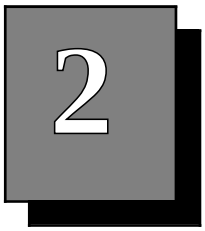
/1/ PROFIBUS Standard, DIN 19245 Part 1, Beuth Verlag GmbH Berlin, April 1991
/2/ PROFIBUS - the process fieldbus standard in industrial communications,
PROFIBUS Nutzerorganisation e.V., Herseler Strasse 31, W-5040 Wesseling, Germany
/3/ MC68302, Integrated Multiprotocol Processor User's Manual, Motorola Inc. 1990
/4/ M68000 Family, Part 1 - Principles and Architecture, te-wi Verlag GmbH Munich
/5/ Documentation PROFIBUS Microcode, Motorola Inc., March 1991
/6/ PROFIBUS - the Fieldbus for Industrial Automation, Carl Hanser Verlag Munich and Vienna
/7/ OS-9 Advanced System Software, Microware Systems Corporation, Iowa, U.S.A
/8/ SMART I/O, [V]IUC, VM30 User's Manual, PEP Modular Computers, W-8950 Kaufbeuren, Germany
/9/ Using OS-9/NET, Microware Systems Corporation, Iowa, U.S.A

 © 1995 PEP Modular Computers

1.3 Ordering Information

Name	Description	Order Number	
OS9PFB-STARTER-II	Starter kit, complete package to support two PROFIBUS nodes	2180-432 /1& 1662	*Old* *New (since Nov. 94)*
PROFI-LIZ-L2+L7	PROFIBUS layer 2 & 7 quantity licenses	2180-432 1666	*Old* *New (since Nov. 94)*
PROFI-LIZ-L2	PROFIBUS layer 2 only licenses	2180-433 1675	*Old* *New (since Nov. 94)*

 © 1995 PEP Modular Computers



2. FUNCTION AND ARCHITECTURE

2.1 Basic Properties

PROFIBUS defines the technical and functional characteristics of a serial fieldbus which interconnects distributed digital field devices in the low (sensor/actuator level) up to the medium (cell level) performance range. The system contains *Master* and *Slave* devices.

A *Master* is able to control the bus, i.e. it may transfer messages without remote request when it has right to access the bus. Masters are called *active stations* in the PROFIBUS protocol. Typical masters are PLCs, CNCs and Cell Controllers.

Slave devices are simple peripheral devices. Typical slaves are sensors, actuators and transmitters. They attain no bus access rights, i.e. they may only acknowledge received messages, or at the request of a master, transmit messages to that master. Slaves are also called *passive stations* in the PROFIBUS protocol. Slaves need only a small part of the protocol and therefore the protocol is especially simple to implement.

The data transmission technique may be adapted to the intended operation area. All variants use the same protocol for medium access and transmission and have the same functions at the interface to the common Application layer.

PROFIBUS includes a powerful layer 7 which contains an optimized interface to layer 2. The logical addressing at the user level enables efficient transmission and fast processing in the end devices.

The PROFIBUS standard defines a comprehensive functionality. Subsets of this functionality are specified in profiles for various application areas.

2.2 Protocol Architecture

PROFIBUS includes definitions for all communication layers of the OSI (Open Systems Interconnection) Reference Model. The architecture of the PROFIBUS protocol is shown in the Figure below.

The *Layers 1 and 2* specify the transmission medium, the physical and electrical properties of the interface, the medium access protocol and the execution of the layer 2 services with their transmission protocols and protocol data units.

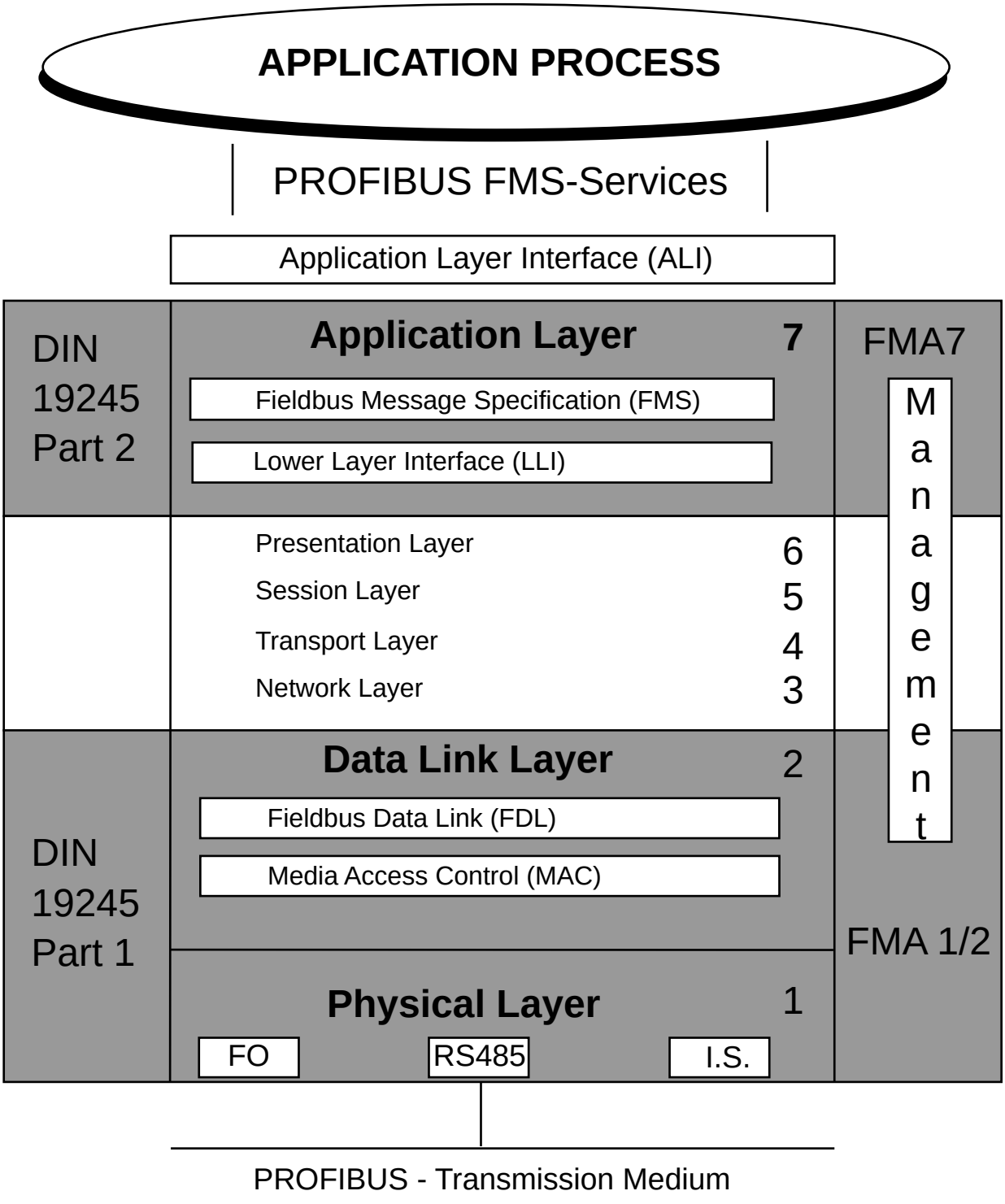
The layers 1 and 2 were published as a pre-standard in DIN V 19 245 Part 1 in 1988. Before the final publication as a standard in December 1990 by the *Deutsche Elektrotechnische Kommision* (DKE), the applicability of the PROFIBUS transmission technique had been substantiated by pilot implementations and extensive tests.

The layers 3 to 6 are not explicit. The functions of these layers that are necessary for the application field of PROFIBUS are combined in the Lower Layer Interface (LLI). The LLI is part of layer 7.

Layer 7 (application protocol) provides the communication functions to the user. They are defined in the *Fieldbus Message Specification* (FMS). FMS realizes the interface to the application process and provides the PROFIBUS user with a variety of powerful application services to access the communication objects of an application process.

 © 1995 PEP Modular Computers

Figure 2.2.0.1: PROFIBUS Protocol Architecture



In addition, the PROFIBUS protocol provides *Network Management* functions.

The functions of layer 7 include a subset of the MMS functions *(MMS, Manufacturing Message Specification)* of the MAP Protocol. The complex functions of MMS are optimized for the requirements at the fieldbus level. Additional fieldbus specific functions for the administration of the communications objects were defined.

 © 1995 PEP Modular Computers

2.3 Layer 1 (Physical Layer)

The area of application of a fieldbus system is substantially effected by the selection of the transmission medium and the physical bus interface. Besides the requirements on the data integrity the costs of provision and installment of the cable are of critical significance.

Hence the PROFIBUS standard defines different versions of the transmission technique under retention of a unique medium access protocol. The RS-485 interface was defined as the base version of the transmission technique.

The US standard fulfills the user requirements on the transmission technique in the areas of discrete part manufacturing, building automation and drive control, as well as in most parts of process control.

In addition to the RS-485 specification, PROFIBUS defines clearly all variable interface parameters, the connector and the bus termination.

The following table defines the basic properties of the RS-485 transmission technique.

Table 2.3.0.1: RS-485 Transmission Technique

Basic Properties of the RS-485 Transmission Technique	
Network topology:	Linear bus, terminated at both ends with the line impedance. Stubs are possible
Medium:	Twisted Pair cable. Shielding may be omitted depending on the application
Number of Stations:	32 Stations without repeaters. When using repeaters extendible to 127 (including 5 repeaters)
Bus length:	max. 1200m, with repeaters extendible up to 4800m depending on the transmission speed
T*ransmission speed:*	9.6 19.2 93.75 187.5 and 500 kbit/s selectable
Connector:	9-Pin D-Sub Connector

 © 1995 PEP Modular Computers

2.4 Layer 2 (Data Link Layer)

2.4.1 Overview

The second layer of the OSI Reference Model realizes the functions of the medium access control and data integrity as well as the execution of the transmission protocols and messages. Layer 2 in PROFIBUS is designated as Fieldbus Data Link (FDL).

The *Medium Access Control (MAC)* defines when a station may transmit data. The MAC has to ensure that only one station has the right to transmit data at any time.

The PROFIBUS protocol has taken two essential requirements for the Medium Access Control into account:

In the case of communication between complex automation components (Masters) with equal rights it has to ensure that each of these stations gets sufficient opportunity to execute its communication tasks within a defined time interval.

In the case of communication between a complex automation device and associated simple peripheral devices (Slaves) it must realize a cyclic real time data exchange as simply as possible.

Therefore, the PROFIBUS medium access protocol includes the *token passing method* for the communication between complex stations (Masters) and additionally the *Master-Slave method* for the communication of the complex stations with the simple peripheral devices (Slaves). This combined method is called *hybrid medium access.*

The *token passing method* ensures, by means of a token, the assignment of the bus access right within a precisely defined time interval. The token message is a special telegram to transfer the right for transmission from one Master to the next Master. It is circulated in a (configurable) maximal token rotation time between all Masters. In the PROFIBUS protocol the token passing method is used only between the complex stations (Masters).

The Master-Slave method allows the Master (active station) that currently owns the right for data transmission to communicate with the associated Slave devices (passive stations). Hereby the Master has the possibility to fetch messages from the Slaves and to transmit messages to the Slaves.

Since in the field area both medium access methods have advantages depending on the application, the hybrid medium access method of PROFIBUS can realize:

• *a pure Master-Slave system*

• *a pure Master-Master system (token passing)*

• *a system with a combination of both methods*

For a certain time after an active station receives the token message it is allowed to exercise the Master function on the bus and communicate with all Slave stations in a Master-Slave communication relationship and with all Master stations in a Master-Master communication relationship.

A token ring means the organizational chain of active stations building a logical ring with their station addresses. In this ring the token, the medium access right, is circulated from one Master to the next Master in a defined sequence (increasing addresses).

In the start-up phase of the bus system the task of the Medium Access Control (MAC) of the active stations is to detect the logical assignment and to establish the token ring. In the operational phase defective or switched-off active stations have to be eliminated from the ring and new active stations have to be included in the ring. These features and also the recognition of defects in the transmission medium and the tranceiver, the detection of errors in the station addressing (e.g. multiple usage) or in the token passing (e.g. multiple token or lost token) are characteristic for the PROFIBUS Medium Access Control.

 © 1995 PEP Modular Computers

Another important task of Layer 2 is data integrity. The PROFIBUS Layer 2 frame formats ensure a high data integrity. All frames have Hamming Distance HD=4. This is achieved by applying the International Standard IEC (International Electrotechnical Commission) 870-5-1 (choice of special start and end delimiters for the telegrams, slip free synchronization, parity bit, control byte ...).

Basically, the PROFIBUS Layer 2 operates connectionless. In addition to the logical peer-to-peer data transmission it provides broadcast and multicast communication.

Broadcast communication means that an active station sends an unconfirmed message to all other stations (Masters and Slaves). Multicast communication means that an active station sends an unconfirmed message to a group of stations (Masters or Slaves).

Layer 2 provides data transmission services to Layer 7. Three services for *acyclic data transmission* and one service for *cyclic transmission* are provided (see Table 2.4.1.1). In addition to the data transmission services layer 2 provides services for Network Management (FMA 1/2).

All layer 2 services are executed at the interface to the LLI through *Service Access Points (SAPs)*. Layer 7 uses these Service Access Points for the addressing of the logical communication relationships. In the active and passive stations multiple SAPs are allowed simultaneously. One distinguishes between source (SSAP) and destination (DSAP) Service Access Points.

Table 2.4.1.1 Data Transmission Services of Layer 2

Data Transmission Services of Layer 2		
***S**end **D**ata With **A**cknowledgement*	(SDA)	acyclic
***S**end and **R**equest **D**ata With Reply*	(SRD)	acyclic
***S**end **D**ata With **N**o Acknowledge*	(SDN)	acyclic
***C**yclic **S**end And **R**equest **D**ata With Reply*	(CSRD)	cyclic

 © 1995 PEP Modular Computers

The technical features of layers 1 and 2, as specified in DIN 19245 Standard Part 1 is shown below.

Table 2.4.1.2 Technical Features of Layers 1 and 2

Technical Features of Layers 1 and 2, DIN 19245 Part 1
Transmission technique corresponding to RS 485, twisted pair, galvanic separation and shielding optional
Further transmission techniques (Fiber optics and Intrinsic Safety) are in preparation
Line length maximal 1200m, with repeaters extendible up to 4800m (depending on the transmission rate) Transmission rate selectable from 9.6 to 500 kbits/s
Total max. 127 stations (active and passive)
NRZ Bit coding (non return to zero)
Asynchronous transmission, half-duplex, slip protected synchronization of the UART characters
Bus access hybrid, combinable decentral and central access
Three acyclic and one cyclic data transmission service
Multi- and broadcast messages and management services
Frame formats according to IEC-870-5-1
Data integrity with Hamming Distance HD=4
Two message priorities

2.4.2 Implementation Overview

The structure of the layer 2 protocol software reflects the division of the layer 2 (Fieldbus Data Link, FDL) in both sub-layers:

• *FLC (Fieldbus Link Control, Transfer Control)*
• *MAC (Medium Access Control, Bus Access Control)*

The FLC sub-layer is accessed via a procedural interface from the layer 2 user. This interface is implemented as a request block interface, therefore the layer 2 user has to provide memory for the telegrams that are to be transferred and for the service parameters. Thereafter the layer 2 user hands over these data structures to the layer 2 by function request or, in the same way, the incoming confirmations or indications can be periodically checked via a cyclic function request.

During requests the FLC sub-layer checks the calling parameter, prepares the telegram and passes it to the MAC sub-layer. During confirmations and indications it accepts the returned telegram and carries out the receipt parameters.

The MAC sub-layer, on the other hand, is completely interrupt controlled. The token protocol is also processed when, for example, the application has entered an endless loop (i.e. as a result of a programming error). The relatively complex access protocol is used as machine condition, where condition changes are triggered using interrupts. The various timers and/or the receipt or transmission of whole or part telegrams act as interrupt sources.

The FLC and MAC sub-layers communicate via various organizational structures contained in memory.

 © 1995 PEP Modular Computers

2.4.3 Resource Circulation

For the hand-over of the service parameters, but most importantly for the storage of the received telegrams, a memory area must be allocated in the form of parameter blocks and telegram buffers. These memory areas are allocated according to the service call and described as resources. The number and order of necessary resources depends upon the service call dynamics, so it would be a waste to hold them permanently in layer 2 since their stored size and order would only be optimized for one particular type of service call and all other calls would make excessive demands upon the memory.

The user of layer 2 (FDL-User) therefore provides these resources on demand and according to the called service.

Request - Confirmation

For the follow-up action (request-confirmation), i.e when the FDL-User has initiated a service request and then awaits a confirmation, a cycle of resources is created. The resources provided during the request remain in the FDL until the receipt of the confirmation and only then are they returned to the FDL-User. Thereafter they are available for use by other services. Typical examples are SDA, SDN and SRD services.

The only time this standard procedure is not applied is when the resources are to remain longer in the FDL, i.e. when they represent the Poll-List or a Service Access Point. In this event there is always a complimentary service that calls these resources back. A typical example is the service pair *LOAD_POLL_LIST* and *DEACT_POLL_LIST* or *ACTIVATE_SAP* and *DEACTIVATE_SAP*.

 © 1995 PEP Modular Computers

The update services take up an intermediate position. Here a pause must be made instead of on the next SRD cycle, whereby the buffer arrives and is then transferred by the update call *n* in the FDL after the confirmation call *n+1* is first sent back.

Indications

No complimentary primitive exists in the layer 2 of the PROFIBUS protocol for primitive service indication in order to stop the resource cycle. A special service call that is not normally specified must therefore be installed in the FDL in order to take care of the necessary indications. The details of the planned services are described in section 4.5 of this manual.

It is the responsibility of the FDL-User to ensure that enough entry buffers and parameter blocks are steadily made available to the FDL in order to work on the received messages. These resources can be transferred on their own or interlinked in packages. The resources must be classified as Service Access Points or Poll-List entries. If no entry buffer is available for a particular Service Access Point or Poll-List entry, then the received telegram is not further worked on.

The classification of entry buffers to Service Access Points or Poll-List entries in the available implementation is SAP referenced. If in a distinct Service Access Point five entry buffers, for example, are transferred then exactly five telegrams can be received. The telegrams are then physically filed away in these five buffers using a copy action.

CSRD - Presentation of the Problem

The actions within layer 2 are normally embedded in unordered service sequences in higher levels. Especially in collaboration with level 7 of the PROFIBUS protocol, the resource circulation through the lower level interface of level 7 is controlled in such a way that no bottlenecks can occur.

Due to the normally bidirectional structure of the communications relationships (request-response), the installment of the transferred requests can be controlled by the installments of the related confirmations, and indirectly through the communications relationship, the installments of the indications.

The CSRD service of layer 2, however, breaksthrough the principals of the resource circulation, in that at the start of the Poll-List a non realizable flood of CSRD confirmations are made from the master and SRD confirmations made from the slave.

It is therefore recommended that in order to take advantage of the normal planned possibilities of the CSRD only SRD cycles containing useful data are worked on .This is possible through the input of "DATA" in the *confirm_mode* parameter of the *LOAD_POLL_LIST* service or in the *indication_mode* parameter of the *FMA2_RSAP_ACTIVATE* service.

 © 1995 PEP Modular Computers

3

3. HARDWARE CONFIGURATION

3.1 PROFIBUS Controllers

The hardware is based on PEP's family of 68302 IUCs (Intelligent Universal Controllers) and CPU modules. Different hardware platforms fulfill the various performance requirements (1.5 - 10 MIPS) and configurations such as VMEbus based systems and busless intelligent I/O nodes.

Table 3.1.0.1: CPU and Controller Characteristics

Product	CPU/ FPU	MIPS Speed (Max)	MIPS Speed (MHz)	CMOS RAM (MByte)	RAM Backup	ROM/ EEPROM (Max)	Serial I/O	MISC Features	Power (W) Typ.	Max. Temp. (˚C)
VM30	68EC030 68882 68302	10	25/40 25/40 16/20	4/8/16/32 DRAM 0.25/1/2 SRAM	VME or Battery	2 MB/ na	2 + 1	Local I/O Extension (CXC)	4.5	-40 to +85
VSBC-4	68302	1.5	16/20	0.5/1 SRAM 2/4 PSRAM*	VME or Battery	2 MB/ 64 KB	2 + 1	Local I/O (CXC)	3.5	-40 to +85
VIUC	68302	1.5	16/20	0.5/1 SRAM 2/4 PSRAM*	VME or Battery	2 MB/ 64 KB	2 + 1	Local I/O Extension (CXC) RTC, Watchdog	3.5	-40 to +85
IUC	68302	1.5	16/20	0.25/0.5/1 SRAM 2/4 PSRAM*	CXC or Battery	1 MB/ 64 KB	2 + 1	Local I/O Extension (CXC) RTC, Watchdog	1.5	-40 to +85
SMART I/O	68302	1.5	20	0.25/0.5/1 SRAM 2/4 PSRAM*	CXC or Battery	1 MB/ 64 KB	2 + 1	Local I/O Extension (CXC) RTC, Watchdog	1.5	-40 to +85

* PSRAM: Pseudo Static RAM

 © 1995 PEP Modular Computers

3.2 The MC68302 IMP (Integrated Multiprotocol Processor)

3.2.1 Overview

The MC68302 contains a 68000 core together with a RISC processor, the latters main objective being the communication e.g. the support of the various assigned protocols.
A short overview of the main function groups that are advantageous in the employment of this controller in the realization of the PROFIBUS protocol is presented below:

System Integration Block (SIB)

- *Independent Direct Memory Access (IDMA)*
- *Interrupt controller with two types of operation*
- *Parallel I/O ports, partly with interrupt generation*
- *2 timers and a watchdog-timer*
- *On-chip 1152 byte dual-port RAM*

Communications Processor (CP)

- *Programmable RISC processor*
- *3 Serial Communication Controllers (SCC 1 - 3)*
- *6 Serial DMA channels for SCC 1-3*
- *SCP for synchronous communication*
- *2 Serial Management Controllers (SMC)*

3.2.2 Microprogramming

The MC68302 RISC processor can run a microprogram that is first loaded into the internal dual-port RAM (DP-Ram). Here the user part of the DP-Ram (576 byte) is available for a microprogram. Motorola Inc. has developed a program that supports the PROFIBUS protocol. The main advantage of the microprogram support is due to the fact that the RISC processor takes over the time consuming actions during the running of the bus protocol. Through this the demand on the 68000 processor is lowered, leaving the application program with more computing power available for other tasks.

3.2.3 Using Internal Function Groups of the MC 68302

The following internal resources of the MC 68302 are used:

Dual-Port-Ram

The PROFIBUS microcode is loaded into the user part of the dual-port RAM. Nothing more is available to the user.

Serial Transmission

The PROFIBUS protocol is realized with the help of one of the three "*Serial Communication Controllers*" (SCC). The cables are:

- *RxD - Receive cable*
- *TxD - Transmission cable*
- *RTS - Switches the RS-485 driver during transmission*

 © 1995 PEP Modular Computers

Timer

In order to operate the PROFIBUS layer 2 protocol software the two internal timers named Timer 1 and 2 are required. If the PROFIBUS layer 7 is employed as well as the PROFIBUS layer 2, a further timer named LLI (Lower Layer Interface of the layer 7) must be prepared. This timer is achieved with the watchdog-timer of the MC 68302.

Interrupt Sources

- *Serial Communication Controller*
- *Timer 1*
- *Timer 2*
- *Timer 3 (LLI timer)*

3.2.4 External Wiring of the MC68302

The demand on external hardware can be minimized due to the PROFIBUS microcode program. The timers used in the realization of the PROFIBUS protocol easily reflect, without exception, the internal timers of the MC 68302. The remaining hardware expenditure limits itself on an *external quartz oscillator running at 24 MHz (9.6 to 500 kBaud).* The setting of the baud rate used is required to have an accuracy of 0.3%.

The output of the quartz oscillator leads to an input "TIN". It serves the baud rate generator as clock input. Furthermore the 68302 module has to be equipped with an RS485 piggyback. Together with the transmission and receive cables of the SCC, the signal "RTS" is required in order to activate the driver components during a transmission request.

 © 1995 PEP Modular Computers

3.3 PROFIBUS 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 encompasses NRZ bit encoding 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 data transfer rates (up to 500 kbit/s).

It is intended to specify further versions, tailored to the following requirements:

- Extended line lengths, line couplers which consume less power and which reduce the influence of defective stations on bus operation, explosive atmosphere protection (Intrinsic Safety) and improved electromagnetic compatibility (possibly with a fibre optic medium).

- Flexible topology, covering a large area (tree topology), applicable for data transfer rates of up to 20 kbit/s, power transmission via the signal conductors, explosive atmosphere protection (Intrinsic Safety).

3.3.1 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 transfer rates ≤ 93.75 kbit/s. For 500 kbit/s the maximum length is reduced to 200m.

Table 3.3.1.1 Electrical Characteristics

Topology	Linear bus, terminated at both ends, stubs ≤ 0.3 m*, no branches
Medium	Shielded Twisted Pair, characteristic impedance between 100 and 130 Ω, minimum conductor area 0.22 mm^2 (24 AWG**), capacity between the conductors about 60 pF/m
Line Length	≤ 1200m, depending on the data transfer rate (cf. EIA RS-485)
Number of Stations	32 (Master stations, Slave stations or repeaters)
Data Transfer Rates	9.6/19.2/93.75 kbits/s for line lengths ≤ 600m, 500 kbit/s for line lengths ≤ 200m
Transceiver Chip	e.g. SN 75176A, DS3695 or others

* **Note**: In contrast to the EIA RS-485 recommendations it is good practice to allow longer stubs if the total of the capacitances of all stubs (Cstges) does not exceed the following values:
Cstges ≤ 0.6 nF @ 500 kbit/s
Cstges ≤ 1.0 nF @ 187.5 kbit/s
Cstges ≤ 3.0 nF @ 93.75 kbit/s
Cstges ≤ 15 nF @ 9.6 and 19.2 kbit/s
It is taken into consideration that the total line length includes the sum of the stub lengths.

** American Wire Gauge

 © 1995 PEP Modular Computers

The dependency of the permissible data transfer rate upon the network expanse (maximum distance between two stations) is shown in Figure A.1 of the US standard EIA RS-422-A (also included in DIN 66259 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 line length and the number of connected stations may be increased by using repeaters (bidirectional amplifiers). A maximum of three 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 AWG 24 twisted pair) is as follows:

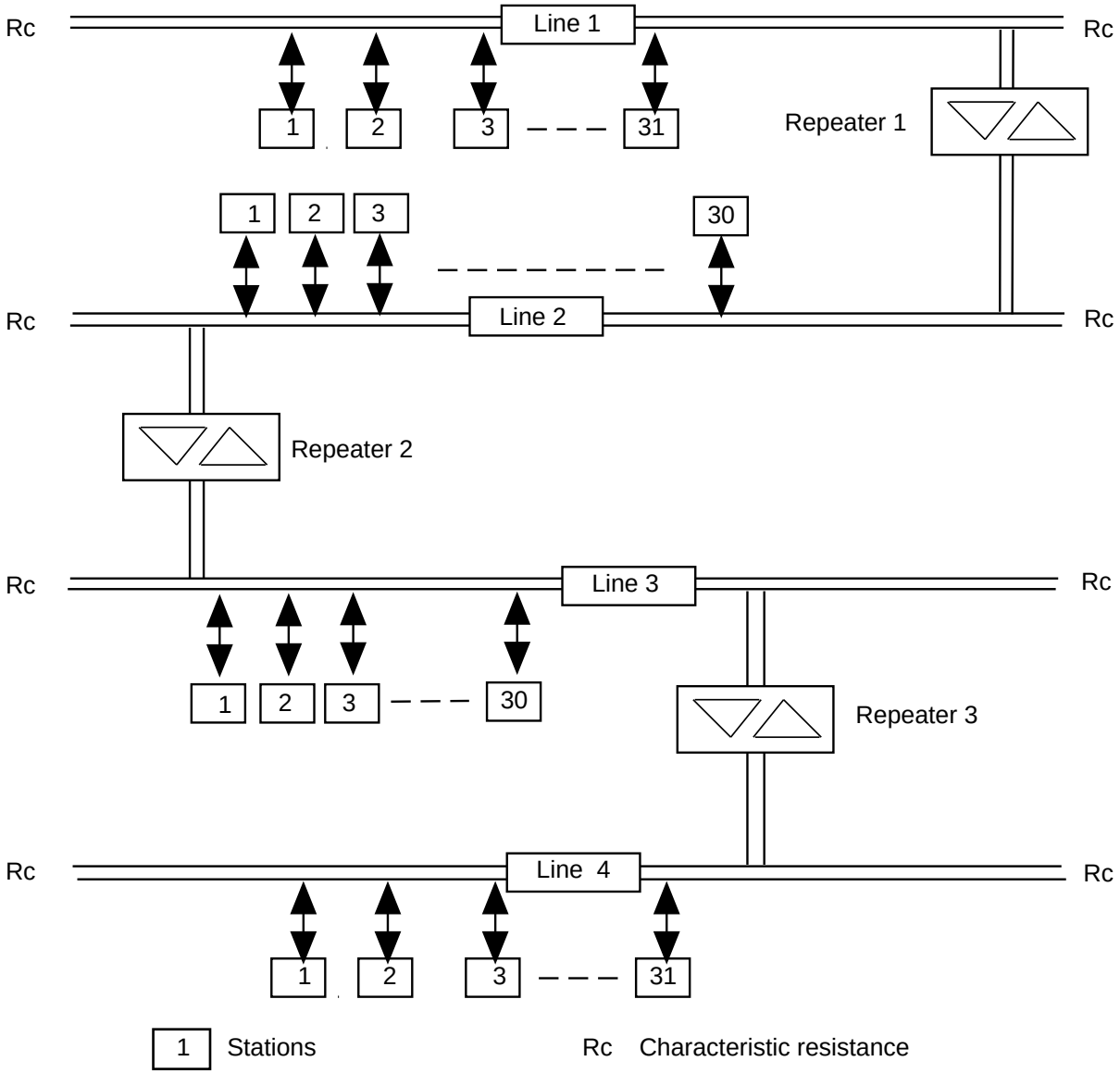
1 repeater: 2.4 km and 62 stations
2 repeaters: 3.6 km and 92 stations
3 repeaters: 4.8 and 122 stations

 © 1995 PEP Modular Computers

Below shows an example of a linear bus topology, with the following characteristics:

93.75 kbit/s	*4 lines, 3 repeaters*
1200m line length	*4800m total length*
30 or 31 stations per line	*122 stations*

Figure 3.3.1.2 Repeater in Linear Bus Topology



 © 1995 PEP Modular Computers

Connector Technique, Mechanical and Electrical Specifications

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 ISO 4902-1980 (DIN 41652, Part 1).

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 realized 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).

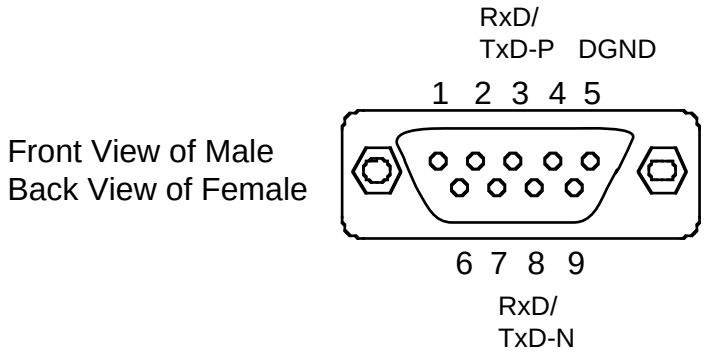
Contact Designations

The pin assignments and layout for the connectors are shown below.

Table 3.3.1.3 Connector Pin Assignments and Layout

Pin No.	RS-485 Ref.	Signal Name	Meaning
1		SHIELD *	Shield, Protective Ground resp.
2		RP *	Reserved for power
3	B/B'	RxD/TxD-P	Receive/Transmit-Data-P
4		CNTR-P*	Control-P
5	C/C'	DGND	Data Ground
6		VP **	Voltage-Plus
7		RP *	Reserved for Power
8	A/A'	RxD/TxD-N	Receive/Transmit-Data-N
9		CNTR-N *	Control-N

* Signal is only necessary at station at end of the bus cable
** Signals are optional



The Data Ground, connected to pin 5, and the Voltage Plus, connected to pin 6, supply the Bus Terminator.

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 (but not mandatory).

The pins 2 and 7 are reserved for separate remote powering of field devices. The definition of signalling and powering related to pins 2, 4, 7 and 9 is not subject to this standard.

 © 1995 PEP Modular Computers

3.4 SC-485F Serial Communications Controller (SCC) Configuration

Information on the configuration of the SC-485F SCC can be found on pages 6-10 of Appendix SCC in this manual.

 © 1995 PEP Modular Computers



4. SOFTWARE ARCHITECTURE

4.1 OS-9 File System and Architecture

Requirements

To run PROFIBUS V3.12 or later, OS-9/PROF V3.0 or later must be installed.

The following files belong to the PROFIBUS layer 2 software in the OS-9/PROFINET directory:

/APPLIC/LAYER_2/OBJS:

`demo`	PROFIBUS Layer 2 application examples that use the PROFIBUS library `pbL2hlf.l`
`demo_M`	
`demo_S`	
`pbmon`	
`pbmode`	
`pbwatch`	

/APPLIC/LAYER_2/SOURCE:

`demo.c`	C-source code of the application examples
`demo_M.c`	
`demo_S.c`	
`pbmode.c`	
`pbmon.c`	
`pbwatch.c`	

/BSP/COMMON/DATMOD:

`busPB.a`	Assembler source of the data module including the PROFIBUS bus parameters. Multiple object files are generated with `makefile` by defining different PROFIBUS devices and different PROFIBUS station numbers
`defsfile`	
`makefile`	Generates multiple object files from `busPB.a`

/BSP/COMMON/NFMDESC:

`n1PROFI.a`	Assembler source of the OS-9/NET device descriptor
`defsfile`	
`makefile`	Generates the object module `n1PROFI`

 © 1995 PEP Modular Computers

/BSP/COMMON/OBJS:

`bPB<p>_<n>`	Data modules containing the PROFIBUS bus parameters for PROFIBUS stations 1,2,3 ..10 (n determines the PROFIBUS station number) and the name of the PROFIBUS device that is used as the interface (p=1 -> /profi_1, p=2 -> /profi_2, p=3 -> /profi_3)
`bPB<p>_I`	Data module with PROFIBUS bus parameters. This module is only usable on an IUC board, because the station number for PROFIBUS is determined by the DIP-switches on the IUC-board
`bPB<p>_S`	Data module with PROFIBUS bus parameters. The PROFIBUS station number is defined by the DIP-switches of one of the CxM status boards STAT-1 or STAT-2 **Note**: Data modules using PROFIBUS device /profi_3 are not provided.
`bPB<p>_M`	Data module with PROFIBUS bus parameters. This module can only be used with a SMART I/O, as the PROFIBUS station number is determined by a value stored in EEPROM.
`phyPROFI`	PROFIBUS Layer 2 (MAC/FLC)
`drvPROFI`	OS-9 driver, interface to PROFIBUS Layer 2
`profiman`	OS-9 manager for PROFIBUS
`n1PROFI`	OS-9/NET device descriptor
`nfPROFI`	OS-9/NET driver accessing PROFIBUS as a medium to transfer data
`n1_nodes`	Ready to use data module for OS-9/NET device n1
`comPROFI`	Communication task linking OS-9/NET driver `nfPROFI` with PROFIBUS

/BSP/DEFS:

`pbL2desc.d`	Definitions to build a PROFIBUS device descriptor
`systype.d`	System definitions

/BSP/SMART/DEFS:

`addr.d`	Definition files for SMART I/O
`vect.d`	

/BSP/SMART/OBJS:

`pSMART_1`	PROFIBUS device descriptor for MC68302 SCC #1 on SMART I/O (default PROFIBUS interface)
`pSMART_2`	PROFIBUS device descriptor for MC68302 SCC #2 on SMART I/O
`pSMART_3`	PROFIBUS device descriptor for MC68302 SCC #3 on SMART I/O

/BSP/SMART/PBL2DESC:

`p_SMART_1.a`	Source file of PROFIBUS device descriptor p_SMART_1
`p_SMART_2.a`	Source file of PROFIBUS device descriptor p_SMART_2
`p_SMART_3.a`	Source file of PROFIBUS device descriptor p_SMART_3
`defsfile`	
`makefile`	

 © 1995 PEP Modular Computers

/BSP/VIUC/DEFS:

`addr.d`	Definition files for (V)IUC
`vect.d`	

/BSP/VIUC/OBJS:

`pVIUC_1`	PROFIBUS device descriptor for MC68302 SCC #1 on (V)IUC
`pVIUC_2`	PROFIBUS device descriptor for MC68302 SCC #2 on (V)IUC
`pVIUC_3`	PROFIBUS device descriptor for MC68302 SCC #3 on (V)IUC

/BSP/VIUC/PBL2DESC:

`pVIUC_1.a`	Source file of PROFIBUS device descriptor pVIUC_1
`pVIUC_2.a`	Source file of PROFIBUS device descriptor pVIUC_2
`pVIUC_3.a`	Source file of PROFIBUS device descriptor pVIUC_3
`defsfile`	
`makefile`	

/BSP/VM30/DEFS:

`addr.d`	Definition files for VM30
`vect.d`	

/BSP/VM30/OBJS:

`pVM30_1`	PROFIBUS device descriptor for MC68302 SCC #1 on VM30
`pVM30_2`	PROFIBUS device descriptor for MC68302 SCC #2 on VM30
`pVM30_3`	PROFIBUS device descriptor for MC68302 SCC #3 on VM30

/BSP/VM30/PBL2DESC:

`pVM30_1.a`	Source file of PROFIBUS device descriptor pVM30_1
`pVM30_2.a`	Source file of PROFIBUS device descriptor pVM30_2
`pVM30_3.a`	Source file of PROFIBUS device descriptor pVM30_3
`defsfile`	
`makefile`	

/CMDS_PEP:

`mksysgo`	Utility to generate a C-source program from a text procedure file

 © 1995 PEP Modular Computers

Definition files for PROFIBUS application programs

/DEFS:

`pbL2con.d` `pbL2type.d`	PROFIBUS definitions in assembler code

`pbL2con.h`
`pbL2type.h`
`pbL2hlf.h`
	PROFIBUS definitions in C-programming language

/LIB:

`pbL2.l`
`pbL2hlf.l`
`pbL2llf.l`

/ROM/SYSGO:

`profigo.txt`	Text procedure file to generate a `sysgo` module for a (V)IUC or VM30 to start OS-9/NET on PROFIBUS automatically
`profigo.c`	C-source of the text file `profigo.txt` generated by the utility `mksysgo`
`viucgo.txt`	Text procedure file to generate a `sysgo` module for a VIUC to start OS-9/RAMNET and OS-9/NET on PROFIBUS automatically
`profigo.c`	C-source of the text file `viucgo.txt` generated by the utility `mksysgo`

/ROM/SMART:

`makefile`	Includes examples to generate OS-9 versions with PROFIBUS for SMART I/O

/ROM/VIUC:

`makefile`	Includes examples to generate OS-9 versions with PROFIBUS for (V)IUC

/ROM/VM30:

`makefile`	Includes an example to generate a romable OS-9 with PROFIBUS for VM30

Note: Files belonging to OS-9/NET can be found on the.*OS-9/PEP_NETPAK* disk.

 © 1995 PEP Modular Computers

OS-9 Modules for PROFIBUS

The following modules must be available in the OS-9 module directory in order to use PROFIBUS:

`phyPROFI`	
`drvPROFI`	
`profiman`	
`profi_<n>`	n = 1, 2 or 3 to determine the PROFIBUS interface port
`busPB`	Neccessary if the application uses the functions of the library `pbL2hlf.l`

OS-9 Modules for OS-9/NET on PROFIBUS

These modules must be loaded into the OS-9 module directory in order to use OS-9/NET on PROFIBUS:

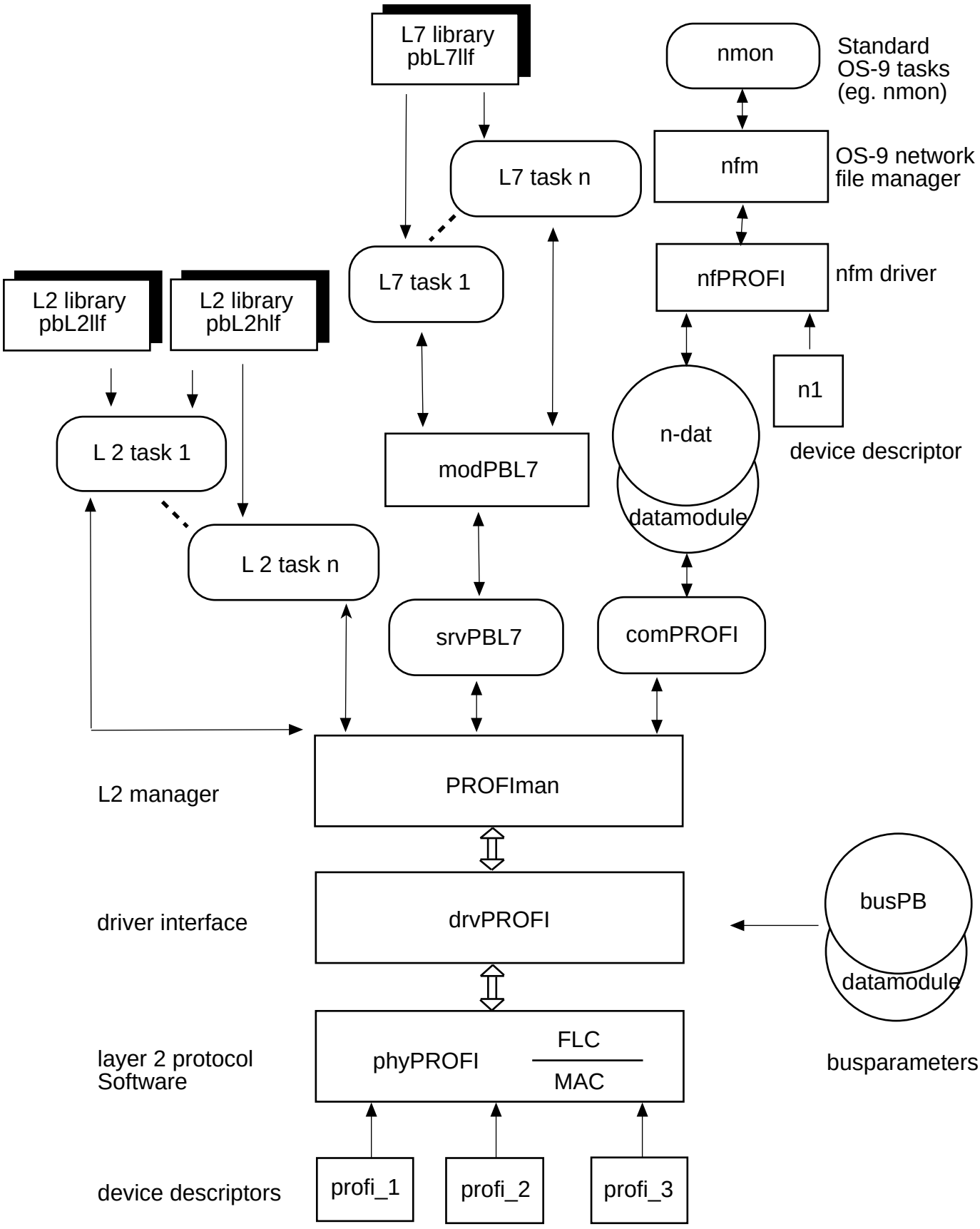
`nfm`	
`nfPROFI`	
`n1`	
`phyPROFI`	
`drvPROFI`	
`profiman`	
`profi_<n>`	n = 1, 2 or 3 to determine the PROFIBUS interface port
`busPB`	
`n1_nodes`	

Note: OS-9/NET for PROFIBUS only works on active PROFIBUS stations. The PROFIBUS local station address may not be 0.

For more information on OS-9/NET for PROFIBUS , refer to reference /9/.

 © 1995 PEP Modular Computers

Figure 4.1.0.1: OS-9 Software Architecture



 © 1995 PEP Modular Computers

4.2 PROFIBUS Installation

The following command sequence is used to copy the files from the PROFIBUS disk to your system disk /dd:

`chd /dm4` if you have a floppy drive connected to a SCSI controller

or

`chd /dm0` if you have a floppy drive connected to the VMSC
`install.PROFINET`

Together with the PROFIBUS files new driver and descriptor object files of the OS-9/PROF V2.4/I2.0 release for the CPU-Boards VM30 and (V)IUC are provided on the PROFIBUS release disk.

Note: Take care that you backup files on your system that are overwritten by the PROFIBUS disk.

4.2.1 Starting PROFIBUS

Under the directory /PROFINET a `makefile` provides the possibility to start PROFIBUS on different CPU types:

```
chd /dd/PROFINET
```

All neccessary files for PROFIBUS are loaded into the module directory. The file `busPB1_1` is selected as module `busPB` to determine the PROFIBUS device and the bus parameters:

PROFIBUS device: */profi_1*
PROFIBUS station address: *1*

OS-9/NET on PROFIBUS is started with the logical station name `PB_1`. This is defined by the data module `n1_nodes`, which provides entries for ten OS-9/NET stations with logical names `PB_1` to `PB_10`.

Additionally when calling `make` the user is able to control which `busPB` module is loaded, to select a different PROFIBUS device and/or a different station address.

For more information type:

```
make
```

 © 1995 PEP Modular Computers

4.2.2 Running PROFIBUS on a VM30 System

```
make pb_VM30
```

This starts the following procedure:

```
load -d ../NET/CMDS/nmon ../NET/CMDS/ndir ../NET/CMDS/nwatch ../NET/CMDS/chp
load -d ../NET/OS9SYS/OBJS/nfm
load -d BSP/COMMON/OBJS/bPB1_1
load -d BSP/COMMON/OBJS/profiman
load -d BSP/COMMON/OBJS/phyPROFI BSP/COMMON/OBJS/drvPROFI
load -d BSP/COMMON/OBJS/nfPROFI BSP/COMMON/OBJS/n1PROFI
              BSP/COMMON/OBJS/n1_nodes
load -d BSP/COMMON/OBJS/comPROFI
load -d BSP/VM30/OBJS/pVM30_1
load -d APPLIC/LAYER_2/OBJS/*
load -d APPLIC/LAYER_7/OBJS/*
nmon /n1 -um &
sleep -s 2
==> include nwatch if you want to monitor the network stations  <==
==> nwatch /n1 -w5 & <==
==> sleep -s 2       <==
tsmon  /pipe/.sh &
```

The PROFIBUS modules for a VM30 are loaded and OS-9/NET is automatically started locally:

PROFIBUS device:	*/profi_1*	*upper port of VM30*
PROFIBUS station address:	*1*	

An example showing how to make a romable OS-9 for a VM30 with PROFIBUS modules is given in */PROFINET/ROM/VM30/makefile*.

 © 1995 PEP Modular Computers

4.2.3 Running PROFIBUS on a VIUC System

```
make pb_VIUC
```

This starts the procedure:

```
load -d ../NET/CMDS/nmon ../NET/CMDS/ndir CMDS/nwatch ../NET/CMDS/chp
load -d ../NET/OS9SYS/OBJS/nfm
load -d BSP/COMMON/OBJS/bPB1_1
load -d BSP/COMMON/OBJS/profiman
load -d BSP/COMMON/OBJS/phyPROFI BSP/COMMON/OBJS/drvPROFI
load -d BSP/COMMON/OBJS/nfPROFI BSP/COMMON/OBJS/n1PROFI
              BSP/COMMON/OBJS/n1_nodes
load -d BSP/COMMON/OBJS/comPROFI
load -d BSP/VIUC/OBJS/pVIUC_1
load -d APPLIC/LAYER_2/OBJS/*
load -d APPLIC/LAYER_7/OBJS/*
nmon /n1 -um &
sleep -s 2
==> include nwatch if you want to monitor the network stations  <==
==> nwatch /n1 -w5 & <==
==> sleep -s 2       <==
tsmon  /pipe/.sh &
```

The PROFIBUS modules for a VIUC are loaded and OS-9/NET is automatically started locally:

PROFIBUS device:	*/profi_1*	*upper port of VIUC*
PROFIBUS station address:	*1*	

An example showing how to make a romable OS-9 for a VIUC with PROFIBUS modules is given in */PROFINET/ROM/VIUC/makefile*.

 © 1995 PEP Modular Computers

4.2.4 Testing the PROFIBUS Connection

The PROFIBUS starter kit board (referred to here as `PB_2`) is connected via a cable that fulfills the DIN 19245 layer 1 requirements to a second PROFIBUS board. This can either be a VIUC or a VM30 in your VMEbus system (referred to here as `PB_1`).

Start PROFIBUS on `PB_1` with the `makefile` under the directory /PROFINET depending on the CPU type:

`make pb_VM30` for a VM30 as the VMEbus Master CPU

`make pb_VIUC` for a VIUC as the VMEbus Master CPU

After the procedure has finished all the necessary PROFIBUS modules are loaded and OS-9/NET on PROFIBUS is started.

After power up on the PROFIBUS starter kit PB_2 a romable OS-9 is brought up and the OS-9/NET on PROFIBUS is automatically started on this station.

The user has to login onto the system with:

```
User name?:    super
Password:      user
```

To test the PROFIBUS connection type in:

```
tmode nopause
ndir -ea /n1
```

The PROFIBUS nodes `PB_1` and `PB_2` are now connected to the network.

Now start a PROFIBUS application on your PROFIBUS nodes `PB_1` and `PB_2`.

1st step: Start the PROFIBUS application on station `PB_2` (= PROFIBUS station number 2).

Type in:

`demo_S 1 10` (*demo_S <remote_station> <sap>*)

This application communicates with the remote PROFIBUS station 1 using Service Access Point 10 for the data transfer. The application prepares a time string with the PROFIBUS service *REPLY_UPDATE* which is picked up from the remote station when an SRD indication occurs. This SRD indication also contains data already sent from station 1.

2nd step: start the corresponding PROFIBUS application on station PB_1 (= PROFIBUS station number 1).

Type in:

`demo_M 2 10` (*demo_M <remote_station> <sap>*)

Each second a time string is transfered to station 2 and at the same time data is picked up from the remote station.

 © 1995 PEP Modular Computers

4.3 Intercommunication Interface

PEP's layer 2 library implements two different ways to establish communications:

- *The PROFIBUS library "pbl2hlf.l"* uses a simplified structure to access the layer 2. The FDL-User does not have to take care of memory management and needs only one function for request and conformation. These services do not include cyclic services.

- *The FDL interface library "pbl2llf.l"* offers the complete functionality of layer 2. The user, however, has to take care of the memory management and the more complex structure of the service calls.

  The service parameters are formed into data blocks which are presented to the FDL, or are received from the FDL accordingly. The necessary memory must be allocated by the FDL-User for these service parameters. In order to reduce the memory requirements dynamic memory allocation is applied as and when required.

  Since the individual FDL- and/or FMA1/2 services contain different quantities and structures of parameters, it stands to reason that no single template can be applied to cover all possibilities. Therefore the parameters for any given service are normally split into several interlinked sub-structures.

  As many of the service routines are not executed immediately, but rather must wait for the correct MAC condition (i.e. token receipt), it often occurs that the installed parameters for a given service routine are installed and remain in the FDL and must be called back with a later service call. Special structures such as bus parameter blocks or parameter blocks used to define the service call address remain in the FDL until layer 2 or the service call address is deactivated.

  The FDL also needs other resources in the form of input buffers and parameter blocks for the evaluation of the incoming messages.

 © 1995 PEP Modular Computers

4.4 PROFIBUS Library "pbl2hlf.l"

This library is a tool for the PROFIBUS user to simplify the use of PROFIBUS services. It provides functions to transfer and receive data to and from a remote station as well as management functions to monitor the PROFIBUS.

The following functions are provided:

General Functions:

open_PROFI
close_PROFI

Data Handling Functions:

open_JOB *(obsolete)*
open_JOB_S
open_JOB_R_SDX
open_JOB_R_SRD
close_JOB
send_SDA
send_SDN
send_SRD
send_RPLUPD_S
send_RPLUPD_M
ready_IND
receive_IND
release_IND

Management Functions:

get_LAS
get_CTR
get_TRR
enable_EVENT
disable_EVENT

 © 1995 PEP Modular Computers

4.4.1 General Functions

open_PROFI

Function:

The PROFIBUS device will be opened. The function determines the name of the PROFIBUS device by the name defined in the data module *busPB*. The service *FMA2_SET_BUSPARAMETER* is executed, the values for the bus parameters are determined by the entries in the *busPB* module. This function must be called before any other function can be used.

C Syntax:

```
USIGN32 open_PROFI ()
```

Return Values:

0 - 126: Station Number
-1: OS-9 system error. Error number is stored in the global variable *errno*

The following return parameters are valid as of Version 3.1 Index 1.3:

0 : no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

close_PROFI

Function:

The PROFIBUS device is closed again. This function should be called before the application terminates.

C Syntax:

```
USIGN32 close_PROFI ()
```

Return Values:

0 - 126: Station Number
-1: OS-9 system error. Error number is stored in the global variable *errno*

The following return parameters are valid as of Version 3.1 Index 1.3:

0 : no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

 © 1995 PEP Modular Computers

4.4.2 Data Transfer Functions

To use data transfer functions the user has to prepare a structure called job descriptor *JOB_DESCR* where information are exchanged between the user and the library functions. Depending on the used function the user has to prepare several entries in the job descriptor and the library function returns information for the user in the job descriptor.

This structure of the job descriptor is defined in the file `pbL2hlf.h`.

```
/* Structure of JOB DESCRIPTOR */

typedef struct JOB_DESCR
{
     USIGN8             job_id;             /* job number                    */
     USIGN8             remote_station;     /* remote station                */
     USIGN8             service;            /* service                       */
     USIGN8             status;             /* status                        */
     USIGN8             ssap;               /* source SAP                    */
     USIGN8             dsap;               /* destination SAP               */
     USIGN8             *send_buf;          /* send buffer for SDA/SDN/SRD   */
                                            /* REPLY UPDATE                  */
     USIGN8             send_len;           /* buffer length                 */
     USIGN8             send_class;         /* priority of data send         */
     USIGN8             *rec_buf;           /* receive buffer for SRD        */
     USIGN8             rec_len;            /* buffer length                 */
     USIGN8             nr_indbuf;          /* number of indication buffer   */
     USIGN8             *ind_buf;           /* indication buffer for         */
                                            /* SDA/SDN/SRD                   */
     USIGN8             ind_len;            /* buffer length                 */
     USIGN8             ind_class;          /* priority of indication data   */
} JOB_DESCR;
```

 © 1995 PEP Modular Computers

open_JOB (Obsolete)

Function:

A job is created for following data transfer actions. The user has to prepare a job descriptor *JOB_DESCR* for further information exchange between the application and the library. The library activates two Service Access Points depending on the value of *ssap* and prepares memory for further data transfer services. This function must be the called before any data transfer function can be executed.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 open_JOB (JOB_DESCR *job_descr)
   JOB_DESCR *job_desrc;
```

Return Values:

0: no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

Job Descriptor:

Entries	>>: Prepared by Application <<: Provided by Library	Value range
job_id	>>	*0...60*
remote_station		
service		
status		
ssap	>>	*0...60*
dsap	>>	*0...60*
**send_buf*		
send_len		
send_class		
**rec_buf*		
rec_len		
nr_indbuf	>>	*0...9*
**ind_buf*		
ind_len		
ind_class		

 © 1995 PEP Modular Computers

job_id: The application is responsible to define the *job_id* number. If several jobs are created, the user has to prepare multiple job descriptors with different *job_id* values.

ssap
dsap By opening a job the application is able to send data packets from a source SAP (Service Access Point) to a destination SAP. Source SAP and destination SAP are defined when the job is created. The value for both can be different.
Note: Internally the library uses two source SAPs (*ssap* and *ssap+1*) so the application has to take care not to use both source SAPs multiple times in different jobs. The value for the receiving station (*remote_station*) can be variable and defined at the time when the data transfer function is called.

nr_indbuf The user has to define the number of buffers that should be provided for PROFIBUS to store incoming indication data. If no indications are expected the value for *nr_indbuf* can be zero.

 © 1995 PEP Modular Computers

open_JOB_S

Function:

A job is created for data transfer actions. The user has to prepare a job descriptor *JOB_DESCR* for further information exchange between the application and the library. The library activates the Service Access Point depending on the value of *ssap* and prepares memory for further data transfer services. This job can be used for SDA/SDN and SRD send requests (*send_SDA*, *send_SDN*, *send_SRD*).

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 open_JOB_S (JOB_DESCR *job_desrc)
```

Return Values:

0: no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

Job Descriptor:

Entries	**>>: Prepared by Application <<: Provided by Library**	**Value range**
job_id	>>	*0..60*
remote_station		
service		
status		
ssap	>>	*0...60*
dsap		
**send_buf*		
send_len		
send_class		
**rec_buf*		
rec_len		
nr_indbuf	>>	*0*
**ind_buf*		
ind_len		
ind_class		

 © 1995 PEP Modular Computers

job_id: The application is responsible to define the *job_id* number. If several jobs are created, the user has to prepare multiple job descriptors with different *job_id* values.

ssap By opening a job the application is able to send data packets from a source SAP (Service Access Point) to a destination SAP. The source SAP is defined when the job is created.
Note: The value for the receiving station (*remote_station*) and the destination SAP (*dsap*) can be variable and defined at the time when the data transfer function is called.

nr_indbuf The value of *nr_indbuf* must be set to zero and no indication can arrive.

The main differences of this function in relation to the *open_JOB* function are listed below:

A job created with *open_JOB* can be used to send request and receive indications.

Sending a request:

send_SDA:	*ssap --> dsap*
send_SDN:	*ssap --> dsap*
send_SRD:	*ssap --> dsap*
send_RPLUPS_S:	*ssap --> dsap*
send_RPLUPD_M:	*ssap --> dsap*

Receiving an indication:

SDA-indication:	*ssap*
SDN-indication:	*ssap*
SRD-indication:	*ssap*

 © 1995 PEP Modular Computers

open_JOB_R_SDX

Function:

A job is created to receive data transfer indications of a remote SDA or SDN request by a *send_SDA* or *send_SDN* function call. The user has to prepare a job descriptor *JOB_DESCR* for further information exchange between the application and the library. The library activates the Service Access Point depending on the value of *ssap* and prepares memory for further data transfer services. This function must be the called before a data indication can be received on that particular SAP.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 open_JOB_R_SDX (JOB_DESCR *job_desrc)
```

Return Values:

0: no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

Job Descriptor:

Entries	**>>: Prepared by Application <<: Provided by Library**	**Value range**
job_id	>>	*0...60*
remote_station		
service		
status		
ssap	>>	*0...60*
dsap		
**send_buf*		
send_len		
send_class		
**rec_buf*		
rec_len		
nr_indbuf	>>	*0...9*
**ind_buf*		
ind_len		
ind_class		

 © 1995 PEP Modular Computers

job_id:	The application is responsible to define the *job_id* number. If several jobs are created, the user has to prepare multiple job descriptors with different *job_id* values.
ssap	By opening a job the application is able to receive data packets on a source SAP (Service Access Point) from a destination SAP. Source SAP is defined when the job is created.
nr_indbuf	The user has to define the number of buffers that should be provided for PROFIBUS to store incoming indication data. If no indications are expected the value for *nr_indbuf* can be zero.

 © 1995 PEP Modular Computers

open_JOB_R_SRD

Function:

A job is created for following data transfer actions. The user has to prepare a job descriptor *JOB_DESCR* for further information exchange between the application and the library. The library activates the Service Access Point depending on the value of *ssap* and prepares memory for further data transfer services. This function must be the called before data transfer function *send_RPLUPD_<x>* can be sent or an indication can be received from a remote Initiator by a *send_SRD* function call.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 open_JOB_R_SDX (JOB_DESCR *job_desrc)
```

Return Values:

0: no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

Job Descriptor:

Entries	**>>: Prepared by Application <<: Provided by Library**	**Value range**
job_id	>>	*0...60*
remote_station		
service		
status		
ssap	>>	*0...60*
dsap		
**send_buf*		
send_len		
send_class		
**rec_buf*		
rec_len		
nr_indbuf	>>	*0...9*
**ind_buf*		
ind_len		
ind_class		

 © 1995 PEP Modular Computers

job_id: The application is responsible to define the *job_id* number. If several jobs are created, the user has to prepare multiple job descriptors with different *job_id* values.

ssap By opening a job the application is able to receive data packets by the *send_RPLUPD_<x>* function call from a source SAP (Service Access Point) to a destination SAP and to receive data packets by a SRD-indication, issued from a remote SRD-request (*send_RPLUPD_<x>*). Source SAP is defined when the job is created.
Note: The value for the receiving station (remote_station) can be variable and defined at the time when the data transfer is called.

nr_indbuf The user has to define the number of buffers that should be provided for PROFIBUS to store incoming indication data. If no indications are expected the value for *nr_indbuf* can be zero.

 © 1995 PEP Modular Computers

close_JOB

Function:

The job is closed. The library deactivates the source SAPs (two source SAPs if opened using *open_JOB*) and the job descriptor is free again.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 close_JOB (USIGN8 job_id)
```

Return Values:

0: no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

Job Descriptor:

Entries	**>>: Prepared by Application <<: Provided by Library**	**Value Range**
job_id	*remains unchanged*	
remote_station	*remains unchanged*	
service	*remains unchanged*	
status	*remains unchanged*	
ssap	*remains unchanged*	
dsap	*remains unchanged*	
**send_buf*	*remains unchanged*	
send_len	*remains unchanged*	
send_class	*remains unchanged*	
**rec_buf*	*remains unchanged*	
rec_len	*remains unchanged*	
nr_indbuf	*remains unchanged*	
**ind_buf*	*remains unchanged*	
ind_len	*remains unchanged*	
ind_class	*remains unchanged*	

 © 1995 PEP Modular Computers

send_SDA

Function:

A data packet is transfered to a destination defined by the destination SAP *dsap* and the station number *remote_station*. For the data transfer the PROFIBUS service SDA is used. The library functions returns to the application, when the SDA confirmation for that SDA request has been passed back from PROFIBUS layer.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 send_SDA (USIGN8 job_id)
```

Return Values:

0: no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

Job Descriptor:

Entries	**>>: Prepared by Application <<: Provided by Library**	**Value Range**
job_id	*remains unchanged*	
remote_station	>>	*0...126*
service		
status		
ssap	*remains unchanged*	
dsap	*remains unchanged or >>*	*0...60*
**send_buf*	>>	*Pointer to send buffer*
send_len	>>	*1...242*
send_class	>>	*HIGH or LOW*
**rec_buf*		
rec_len		
nr_indbuf	*remains unchanged*	
**ind_buf*		
ind_len		
ind_class		

dsap	Job created by *open_JOB*:	remains unchanged.
	Job created by *open_JOB_S*:	must be now defined and can be variable.

send_buf **Note**: The application has to prepare the send buffer. The real user data written to the send buffer have to start at the 12th byte. Bytes 0- 11 are reserved for PROFIBUS and should not be touched.

send_len Length of real user data to send.

Note: This function can be used only on active stations

 © 1995 PEP Modular Computers

send_SDN

Function:

A data packet is transfer to a destination defined by the destination SAP *dsap* and the station number *remote_station*. For the data transfer the PROFIBUS service SDN is used. The library functions returns to the application, when the SDN confirmation for that SDN request has been passed back from PROFIBUS layer.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 send_SDN (USIGN8 job_id)
```

Return Values:

0: no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

Job Descriptor:

Entries	**>>: Prepared by Application <<: Provided by Library**	**Value Range**
job_id	*remains unchanged*	
remote_station	>>	*0...126, or global address 127*
service		
status		
ssap	*remains unchanged*	
dsap	*remains unchanged or >>*	*0…60*
**send_buf*	>>	*Pointer to send buffer*
send_len	>>	*1…242*
send_class	>>	*HIGH or LOW*
**rec_buf*		
rec_len		
nr_indbuf	*remains unchanged*	
**ind_buf*		
ind_len		
ind_class		

dsap	Job created by *open_JOB*: Job created by *open_JOB_S*:	remains unchanged. must be now defined and can be variable.
send_buf:	**Note**: The application has to prepare the send buffer. The real user data written to the send buffer have to start at the 12th byte. Bytes 0- 11 are reserved for PROFIBUS and should not be touched.	
send_len	Length of real user data to send.	

Note: This function can be used only on active stations.

 © 1995 PEP Modular Computers

send_SRD

Function:

A data packet is transfered to a destination defined by the destination SAP 'dsap' and the station number 'remote_station'. If the destination has prepared data via the *REPLY_UPDATE* service these data are passed to the application. For the data transfer the PROFIBUS service SRD is used. The library functions returns to the application, when the SRD confirmation for that SRD request has been passed back from PROFIBUS layer.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 send_SRD (USIGN8 job_id)
```

Return Values:

0: no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

Job Descriptor:

Entries	**>>: Prepared by Application <<: Provided by Library**	**Value Range**
job_id	*remains unchanged*	
remote_station	>>	*0...126*
service		
status		
ssap	*remains unchanged*	
dsap	*remains unchanged or* >>	*0...60*
**send_buf*	>>	*Pointer to send buffer*
send_len	>>	*1...242*
send_class	>>	*HIGH or LOW*
**rec_buf*	<<	*Pointer to receive buffer*
rec_len	<<	*0...242*
nr_indbuf	*remains unchanged*	
**ind_buf*		
ind_len		
ind_class		

dsap	Job created by *open_JOB*:	remains unchanged.
	Job created by *open_JOB_S*:	must be now defined and can be variable.

send_buf **Note**: The application has to prepare the send buffer. The real user data written to the send buffer have to start at the 12th byte. Bytes 0- 11 are reserved for PROFIBUS and should not be touched.

send_len Length of real user data to send.

 © 1995 PEP Modular Computers

rec_buf Points to the buffer where the received user data is located. This buffer is provided by the library and can be overwritten by the next use of the *send_SRD* function.

rec_len Length of received user data.

Note: This function can be used only on active stations.

 © 1995 PEP Modular Computers

send_RPLUPD_S

Function:

A data packet is transfered to a destination defined by the destination SAP *dsap* and the station number *remote_station*. For the data transfer the PROFIBUS service *REPLY_UPDATE* in single mode is used. The library functions returns to the application, when the request has been completed by the confirmation. The data is sent when an SRD request from a remote station has been performed.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 send_RPLUPD_S (USIGN8 job_id)
```

Return Values:

0: no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

Job Descriptor:

Entries	**>>: Prepared by Application** **<<: Provided by Library**	**Value Range**
job_id	*remains unchanged*	
remote_station	*>>*	*0...126*
service		
status		
ssap	*remains unchanged*	
dsap	*remains unchanged or >>*	*0..60*
**send_buf*	*>>*	*Pointer to send buffer*
send_len	*>>*	*1...242*
send_class	*>>*	*HIGH or LOW*
**rec_buf*		
rec_len		
nr_indbuf	*remains unchanged*	
**ind_buf*		
ind_len		
ind_class		

dsap	Job created by *open_JOB*:	remains unchanged.
	Job created by *open_JOB_R_SRD*:	must be now defined and can be variable.
send_buf	**Note**: The application has to prepare the send buffer. The real user data written to the send buffer have to start at the 12th byte. Bytes 0- 11 are reserved for PROFIBUS and should not be touched.	
send_len	Length of real user data to send.	

 © 1995 PEP Modular Computers

send_RPLUPD_M

Function:

A data packet is transfered to a destination defined by the destination SAP *dsap* and the station number *remote_station*. For the data transfer the PROFIBUS service *REPLY_UPDATE* in multiple mode is used. The library functions returns to the application, when the request has been completed by the confirmation. The user is informed about the data transfer by a SRD indication. The requested data remains available until it is overwritten, thus enabling a multiple readout of this data with every SRD request from a remote station.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 send_RPLUPD_M (USIGN8 job_id)
```

Return Values:

0: no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

Job Descriptor:

Entries	**>>: Prepared by Application** **<<: Provided by Library**	**Value Range**
job_id	*remains unchanged*	
remote_station	>>	*0...126*
service		
status		
ssap	*remains unchanged*	
dsap	*remains unchanged or >>*	*0...60*
**send_buf*	>>	*Pointer to send buffer*
send_len	>>	*1...242*
send_class	>>	*HIGH or LOW*
**rec_buf*		
rec_len		
nr_indbuf	*remains unchanged*	
**ind_buf*		
ind_len		
ind_class		

dsap	Job created by *open_JOB*:	remains unchanged.
	Job created by *open_JOB_R_SRD*:	must be now defined and can be variable.
send_buf	**Note**: The application has to prepare the send buffer. The real user data written to the send buffer have to start at the 12th byte. Bytes 0- 11 are reserved for PROFIBUS and should not be touched.	
send_len	Length of real user data to send.	

 © 1995 PEP Modular Computers

ready_IND

Function:

This function informs the user if any indication is available due to SDA, SDN or SRD requests from a remote station or an EVENT indication if the function *enable_EVENT* has been used.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 ready_IND ()
```

Return Values:

0: no indication available
1: indication available
-1: OS-9 system error. Error number is stored in the global variable *errno*

receive_IND

Function:

Waits for an indication. This can be a SDA, SDN, SRD or EVENT indication. To get a SDA, SDN or SRD indication a job has to be created via the functions *open_JOB_S*, *open_JOB_R_SDX* or *open_JOB_R_SRD*. An EVENT indication can occur, if the function *enable_EVENT* has been executed. Using this function, the application is locked in the library till an indication has occurred.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 receive_IND ()
```

Return Values:

0-60: *job_id* of corresponding job descriptor, where the information of an SDA, SDN or SRD indication is stored by the library
255 an EVENT indication has occurred, a job descriptor is not relevant
-1: OS-9 system error. Error number is stored in the global variable *errno*

 © 1995 PEP Modular Computers

Job Descriptor (SDA, SDN or SRD Indication)

Entries	**>>: Prepared by Application <<: provided by library**	**Value Range**
job_id	*remains unchanged*	
remote_station	<<	*0...126, or global address 127*
service	<<	*SDA, SDN, SRD*
status	<<	*for SDA, SDN: OK* *for SRD: NO, LO, HI*
ssap	*remains unchanged*	
dsap	*remains unchanged*	
**send_buf*		
send_len		
send_class		
**rec_buf*		
rec_len		
nr_indbuf	*remains unchanged*	
**ind_buf*	<<	*pointer to indication buffer*
ind_len	<<	*1...242*
ind_class	<<	*HIGH or LOW*

remote_station Indicates from where the data has been send

service Determines the service type of the indication

status The status of the indication.
For status value explanations, please refer to Appendix A.

ind_buf Points to the buffer where the received user data are located. This buffer is provided by the library and can be overwritten by the next use of the *receive_IND* function.

ind_len Length of received user data.

 © 1995 PEP Modular Computers

release_IND

Function:

This function releases again the indication buffer, in order that the contents of the indication buffer can be overwritten without further notice. Therefore the user has to read the buffer before he releases it. This function must be used after any received SDA, SDN or SRD indication.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 release_IND (USIGN8 job_id)
```

Return Values:

0:	No error
-1:	OS-9 system error. Error number is stored in the global variable *errno*
else:	PROFIBUS status value. For status value explanations, refer to Appendix A.

Note: The application examples `demo.c`, `demo_M.c` and `demo_S.c` under the directory /PROFINET/APPLIC/LAYER_2/SOURCE gives advice how to use the library functions for data transfer.

 © 1995 PEP Modular Computers

4.4.3 Management Functions

The following functions provides information for the application in order to monitor the PROFIBUS protocol.

get_LAS

Function:

Returns the list of active stations on the PROFIBUS. The application has to prepare a 128-byte buffer. The pointer to the buffer is passed to the library function, where the buffer is updated.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 get_LAS (USIGN8 *buffer)
```

Return Values:

0: no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

Each entry of the buffer field reflects the status of the corresponding station number:

0x00: station is not active in the logical token ring
0x01: station is active in the logical token ring

Note: This function can be used only on active stations

 © 1995 PEP Modular Computers

get_CTR

Function:

Returns statistic values. The application has to prepare a 4-long word buffer. The pointer to the buffer is passed to the library function, where the buffer is updated.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 get_CTR (USIGN32 *buffer)
```

Return Values:

0: no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

Each entry of the buffer field is updated with statistic information:

buffer[0]: number of sent telegrams
buffer[1]: number of repeated telegrams
buffer[2]: number of correct start delimiters
buffer[3]: number of defective start delimiters

 © 1995 PEP Modular Computers

get_TRR

Function:

Returns the Real Target Rotation Time. The application has to prepare a 1-long word buffer. The pointer to the buffer is passed to the library function, where the buffer is updated.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 get_TRR (USIGN32 *buffer)
```

Return Values:

0: no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

The buffer field is updated with the value of the Real Target Rotation Time.

Note: This function can be used only on active stations

 © 1995 PEP Modular Computers

enable_EVENT

Function:

This function enables the receipt of FMA2 event or error indications. The application has to prepare a 1-byte buffer. The pointer to the buffer is passed to the library function, where the buffer is updated when an FMA2 event occurs.

C Syntax:

```
#include <pbL2hlf.h>

USIGN32 enable_EVENT (USIGN8 *buffer)
```

Return Values:

0: no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

The application has to use the function *receive_IND* to get informed when an EVENT indication occurs. The buffer is then updated with the event or error status number.

Status Values:

0x01 (FMA2_FAULT_ADDRESS):	Multiple FDL addresses
0x02 (FMA2_FAULT_TRANSCEIVER):	Error in transmitter or receiver
0x03 (FMA2_FAULT_TTO):	Bus timeout
0x04 (FMA2_FAULT_SYN):	No receiving synchronization
0x05 (FMA2_FAULT_OUT_OF_RING):	Active station has left the logical token ring
0x06 (FMA2_GAP_EVENT):	A new station has been inserted into the GAP area

 © 1995 PEP Modular Computers

disable_EVENT

Function:

This function disables again the receipt of FMA2 event or error indications.

C Syntax:

```
#include <pbL2hlf.h>

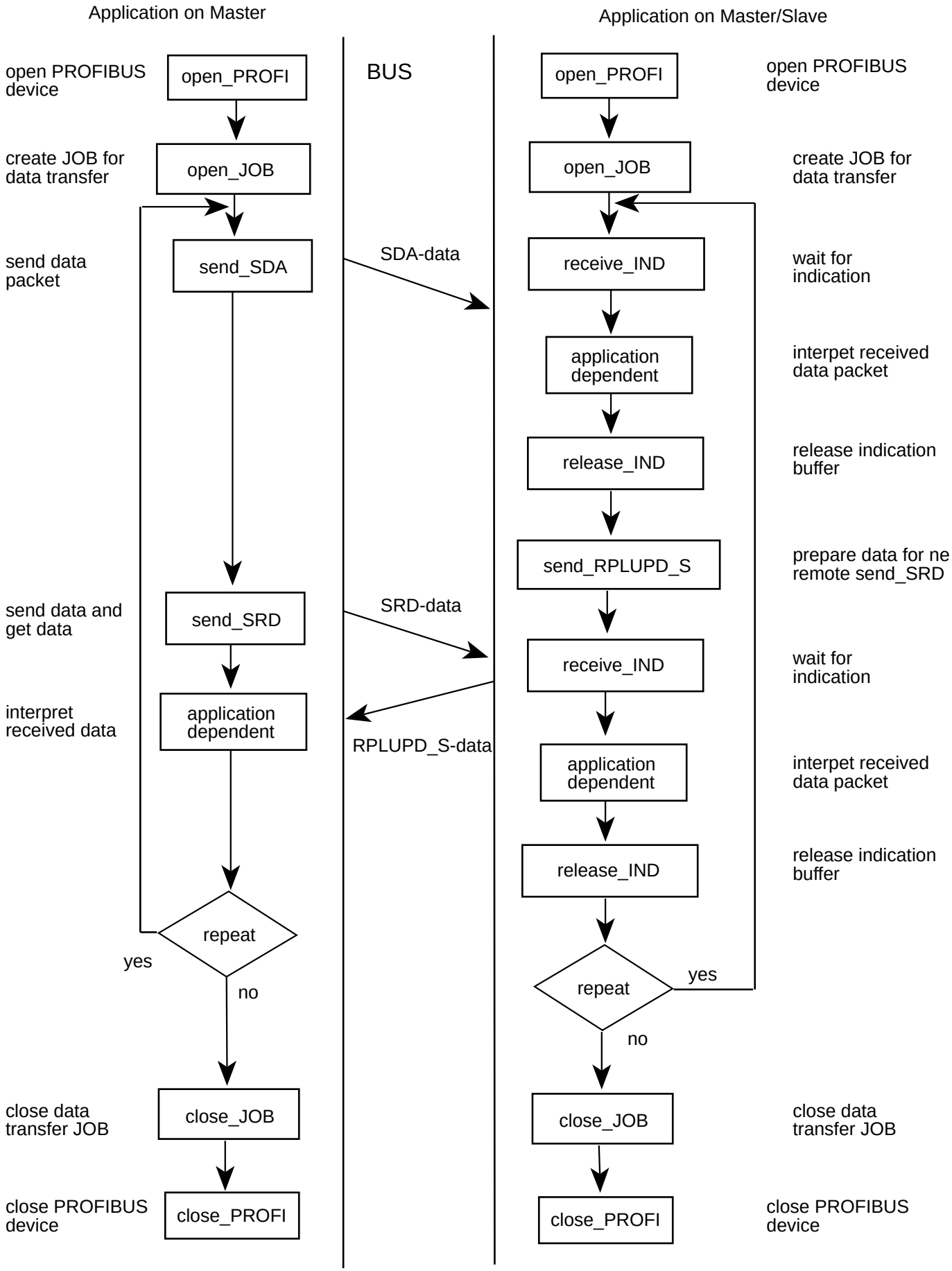
USIGN32 disable_EVENT ()
```

Return Values:

0: no error
-1: OS-9 system error. Error number is stored in the global variable *errno*
else: PROFIBUS status value. For status value explanations, refer to Appendix A.

Note: The application examples `pbmon.c` and `pbwatch.c` under the directory /PROFINET/APPLIC/LAYER_2/SOURCE gives advice on how to use the management library functions.

An application program structure using the `pbl2hlf.l` library is shown overleaf.



 © 1995 PEP Modular Computers

4.5 FDL Interface Library "pbl2llf.l"

A standard OS-9 structure is used with manager, driver and descriptors to realize an interface between PROFIBUS layer 2 applications (FDL-User) and the PROFIBUS layer 2 protocol (FDL) itself. The user is able to setup FDL layer 2 services via I/O-functions in the C language.

The C library `pbL2llf.l` is provided to ease the communication between the FDL-User and the FDL via the PROFIBUS interface driver.

The communication is achieved via five interface functions:

```
fdl_open ()
fdl_req ()
fdl_con_ind ()
fdl_con_ind_poll()
fdl_close
```

4.5.1 Function fdl_open

```
int fdl_open
    (char * device_name)
```

Return Values
-1: error
0: OK

This function initializes the PROFIBUS device. It requires a pointer to the PROFIBUS device name (i.e */profi_1*) and must be used by the FDL-User before any communication with the FDL can take place.

4.5.2 Function fdl_req

```
int fdl_req
    (T_FDL_SERVICE_DESCR *sdb_ptr)
```

Return Values
-1: error
0: OK

This function is used to implement requests and provide the FDL with a pointer to the Service Description Block of type *T_FDL_SERVICE_DESCR* which contains the occurring parameter and a pointer to the service specific parameter blocks for any given service.

4.5.3 Function fdl_con_ind

```
T_FDL_SERVICE_DESCR * fdl_con_ind (void)
```

Return Values
-1: error
else: pointer to the Service Description Block

This function allows the FDL-User to distinguish between confirmation and indications. The result of this function forms a pointer to the Service Description Block of type *T_FDL_SERVICE_DESCR* which contains the service variable parameters and gives direction to the service specific parameter blocks. The function only returns to the application program when a confirmation or indication arrives.

 © 1995 PEP Modular Computers

4.5.4 Function fdl_con_ind_poll

```
T_FDL_SERVICE_DESCR * fdl_con_ind_poll (void)
```

This function allows the FDL-User to distinguish between confirmation and indications. The result of this function forms a pointer to the Service Description Block of type *T_FDL_SERVICE_DESCR* which contains the service variable parameters and gives direction to the service specific parameter blocks. If no confirmation or indication is available a NULL-pointer is returned.

Return Values

-1:	error
0:	no confirmation or indication available
else:	pointer to the Service Description Block

4.5.5 Function fdl_close

```
int fdl_close (void)
```

This function is used to terminate the communication to the FDL, leading to the PROFIBUS device being closed.

Return Values:

-1:	error
0:	OK

4.5.6 FDL Services

The FDL (Fieldbus Data Link) services are made available to the user via layer 2. The following data transfer services are available:

- *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)*

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 the 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.

To simplify the overview of the intercommunication interface, some terms have been selected which differ slightly from those in the normal specification.

A list of the FDL services as defined in DIN 19245, Part 1 follows below. On the right hand side two columns form constants "*service*" and "*primitive*" as they must be given in the Service Description Block (*T_FDL_SERVICE_DESCR*). These terms are agreed for use in the include data `pbL2con.h`

A deviation from the DIN 19245, Part 1 standard occurs with the CSRD service; according to standards, acknowledgment of the first CSRD.confirmation is made after a CSRD.request has loaded a Poll-List. All further CSRD.confirmations show the completion of each SRD cycle and therefore take on the character of a CSRD.confirmation but have a different meaning than the first confirmation.

Thought has been given to this condition, and as soon as the Poll-List has been loaded (*LOAD_POLL_LIST*) and is confirmed, CSRD.con is used thereafter to confirm the completion of individual poll cycles. A CSRD.req service does not exist.

 © 1995 PEP Modular Computers

Terminology to DIN 19245, Part 1	Intercommunication Interface		
	Service	Primitive	Possible for
Send Data with Acknowledge (SDA)			
FDL_DATA_ACK.request	*SDA*	*REQ*	*M*
FDL_DATA_ACK.confirm	*SDA*	*CON*	*M*
FDL_DATA_ACK.indication	*SDA*	*IND*	*M and S*
Send Data with No Acknowledge (SDN)			
FDL_DATA.request	*SDN*	*REQ*	*M*
FDL_DATA.confirm	*SDN*	*CON*	*M*
FDL_DATA.indication	*SDN*	*IND*	*M and S*
Send and Request Data with Reply (SRD)			
FDL_DATA_REPLY.request	*SRD*	*REQ*	*M*
FDL_DATA_REPLY.confirm	*SRD*	*CON*	*M*
FDL_DATA_REPLY.indication	*SRD*	*IND*	*M and S*
FDL_REPLY_UPDATE.request	*REPLY_UPDATE*	*REQ*	*M and S*
FDL_REPLY_UPDATE.confirm	*REPLY_UPDATE*	*CON*	*M and S*
Cyclic Send and Request Data with Reply (CSRD)			
FDL_SEND_UPDATE.request	*SEND_UPDATE*	*REQ*	*M*
FDL_SEND_UPDATE.confirm	*SEND_UPDATE*	*CON*	*M*
FDL_CYC_DATA_REPLY.request	*LOAD_POLL_LIST*	*REQ*	*M*
FDL_CYC_DATA_REPLY.confirm (1st confirmation)	*LOAD_POLL_LIST*	*CON*	*M*
FDL_CYC_DATA_REPLY.confirm (2nd upwards confirmation)	*CSRD*	*CON*	*M*
FDL_CYC_ENTRY.request	*POLL_ENTRY*	*REQ*	*M*
FDL_CYC_ENTRY.confirm	*POLL_ENTRY*	*CON*	*M*
FDL_CYC_DEACT.request	*DEACT_POLL_LIST*	*REQ*	*M*
FDL_CYC_DEACT.confirm	*DEACT_POLL_LIST*	*CON*	*M*

M: Master
S: Slave

Brief descriptions of each of the data transfer services follow below with the following notation being used in the Figures:

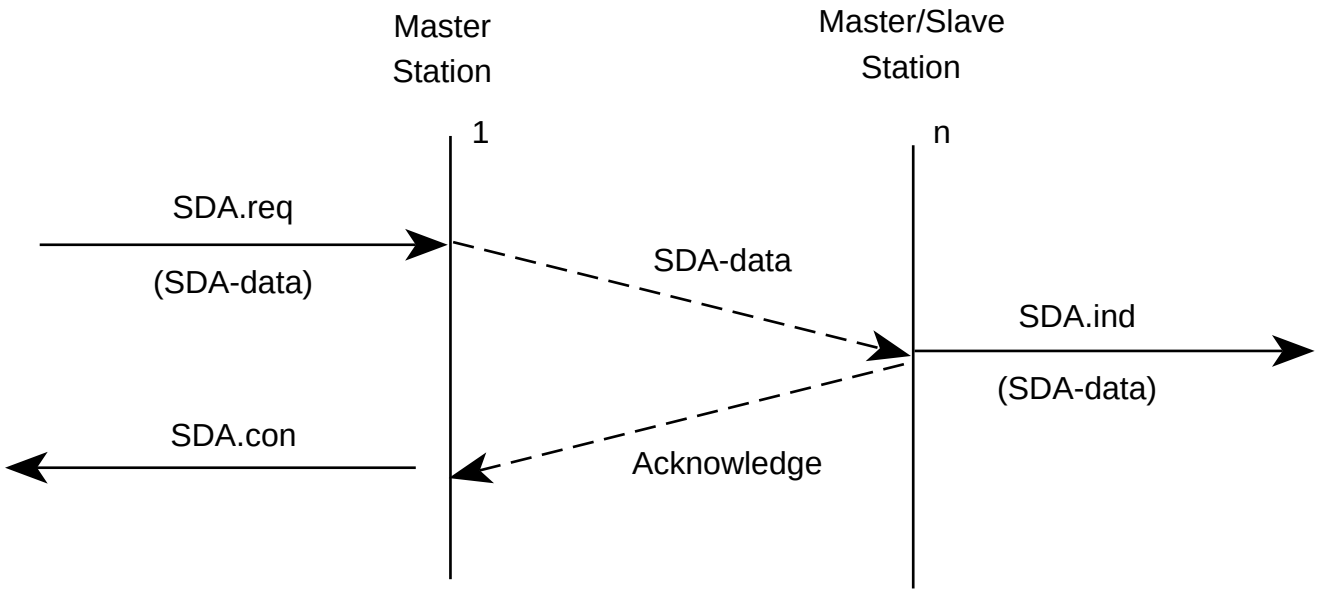
.req .request
.ind .indication
.con .confirmation

 © 1995 PEP Modular Computers

Send Data with Acknowledge (SDA)

This service allows a user of the FDL (Layer 2) in a Master station (referred to as a Local User), to send user data (SDA-data) to a single remote station. At the remote station the SDA-data, if received error-free, is delivered by the FDL to the user (referred to as a Remote User). 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 repeats the data transfer.

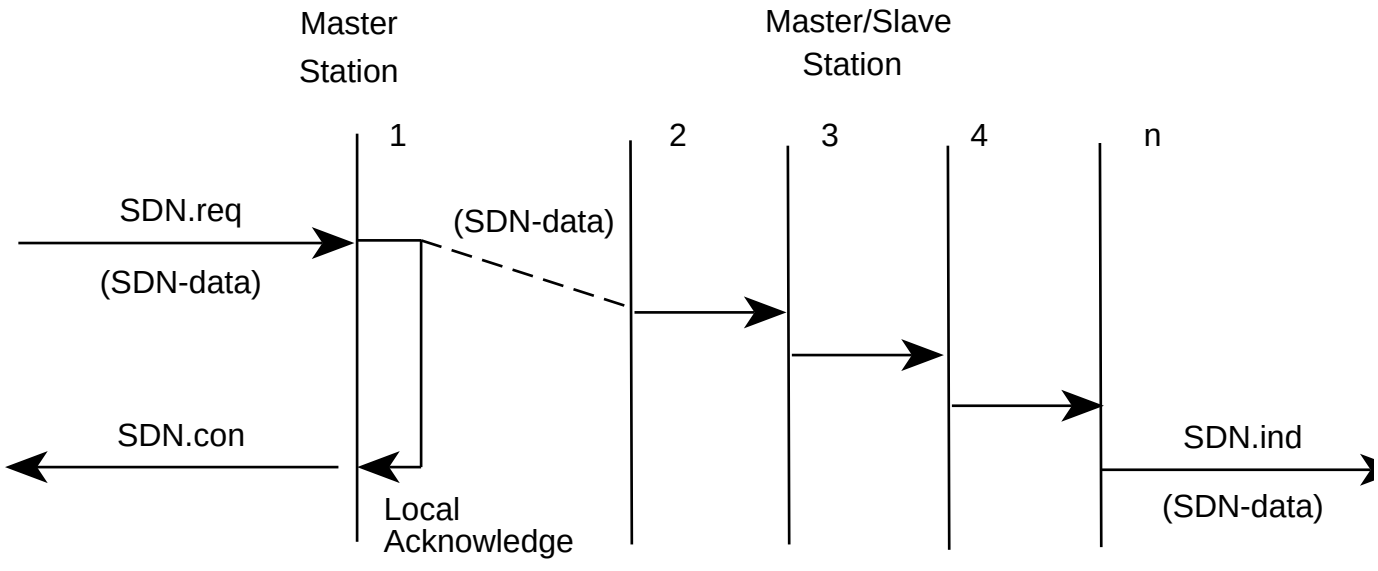
Figure 4.5.6.1: SDA Service



Send Data with No Acknowledge (SDN)

This service allows a Local User to transfer data (SDN-data) 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 SDN-data, if received error-free, is passed to the Remote User. There is no confirmation, however, that such a transfer has taken place.

Figure 4.5.6.2: SDN Service



 © 1995 PEP Modular Computers

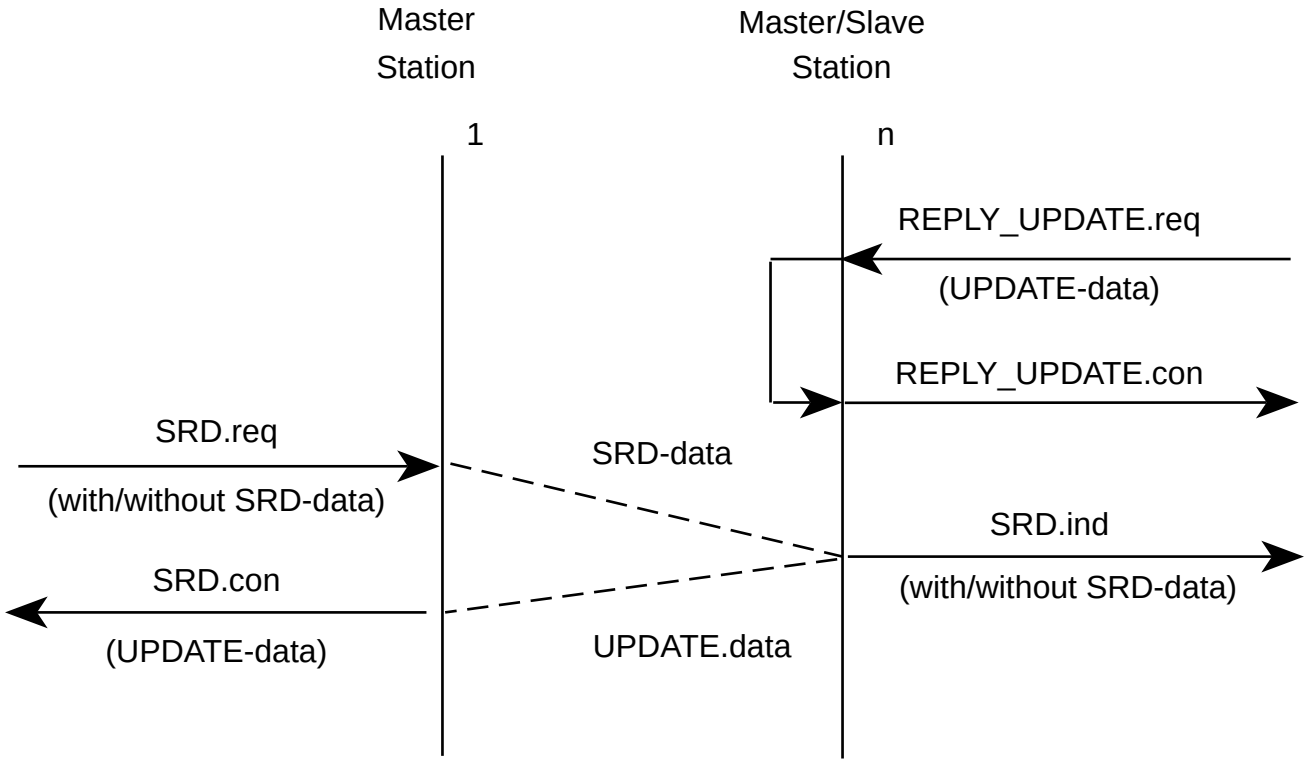
Send and Request Data with Reply (SRD)

This service allows a Local User to transfer data (SRD-data) to a single remote station and at the same time to request data (UPDATE-data) that was made available by the Remote User at an earlier time. At the remote station the received SRD-data, 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 (SRD-data=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 transfered data.

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

Figure 4.5.6.3: SRD Service



Cyclic Send and Request Data with Reply (CSRD)

This service allows a Local User to cyclically transfer data (S_UPDATE-data) to a remote station and at the same time to request data (R_UPDATE-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.

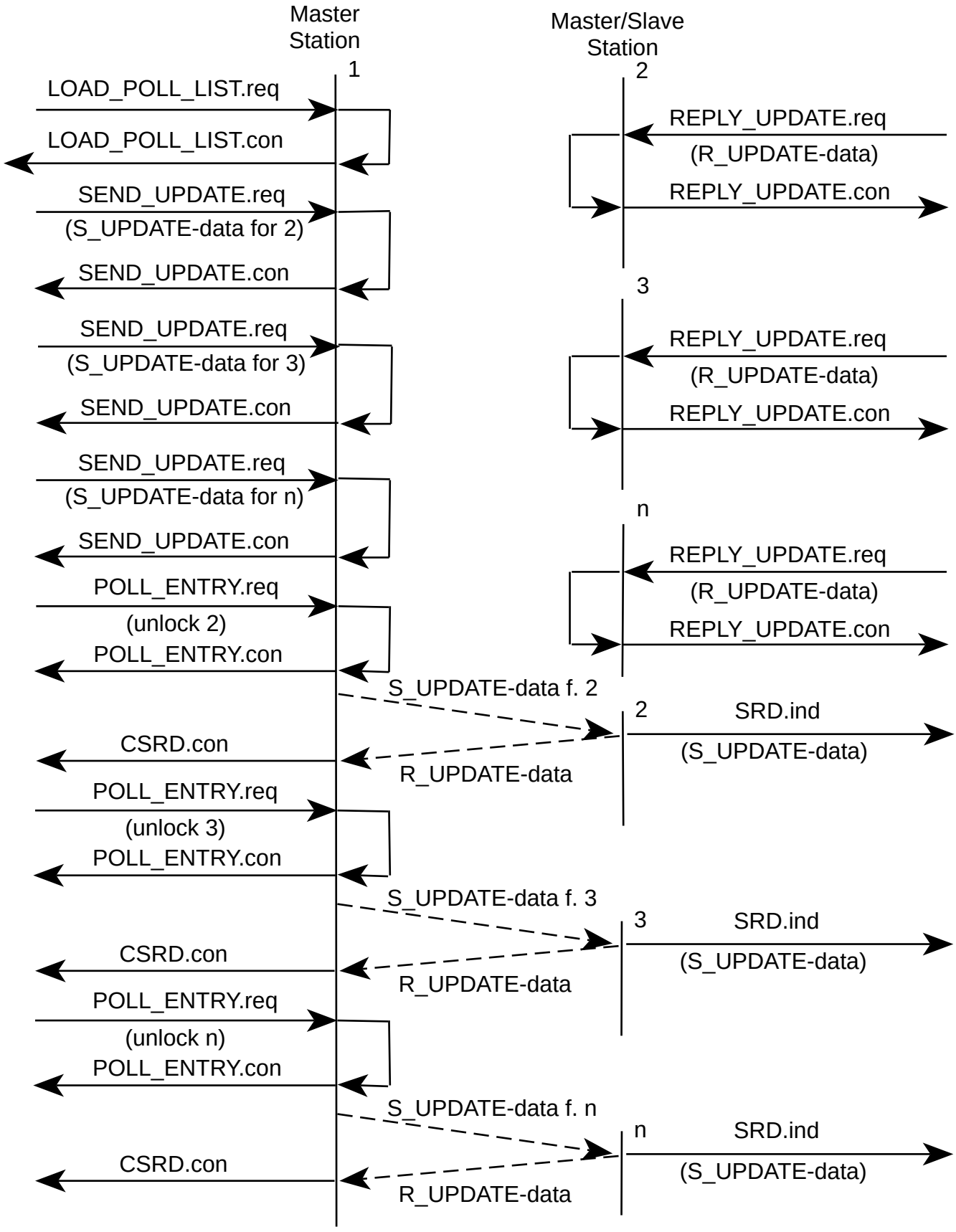
The Local User cyclically receives either the requested data or an indication that the data was not available or a confirmation of the non-receipt of the transmitted data. The first two reactions also confirm the receipt of the transfered data.

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 is defined by the Local User in the Poll-List.

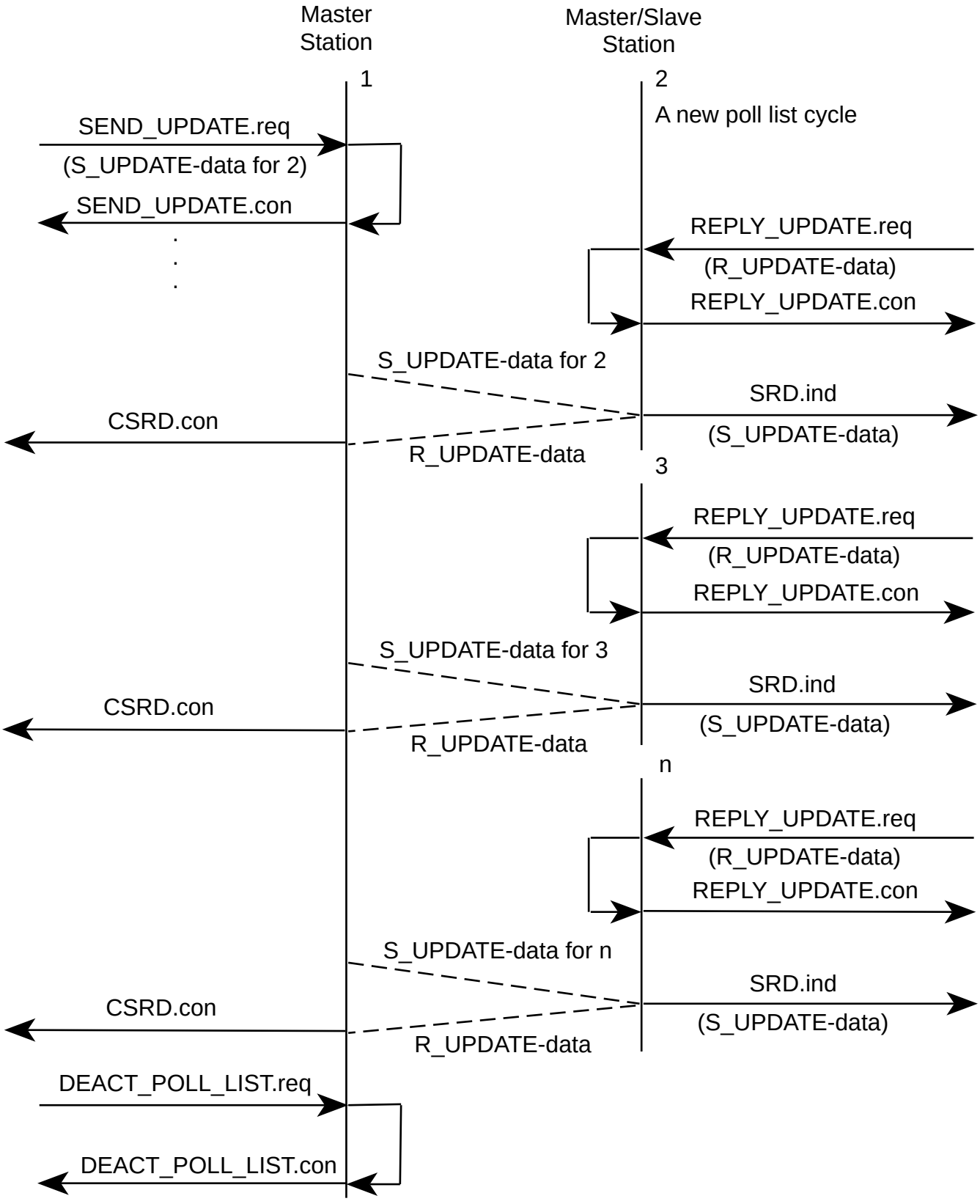
 © 1995 PEP Modular Computers

Figure 4.5.6.4 Start of CSRD Service



 © 1995 PEP Modular Computers

Figure 4.5.6.5 End of CSRD Service



Individual descriptions of the layer 2 services are now described on the following pages.

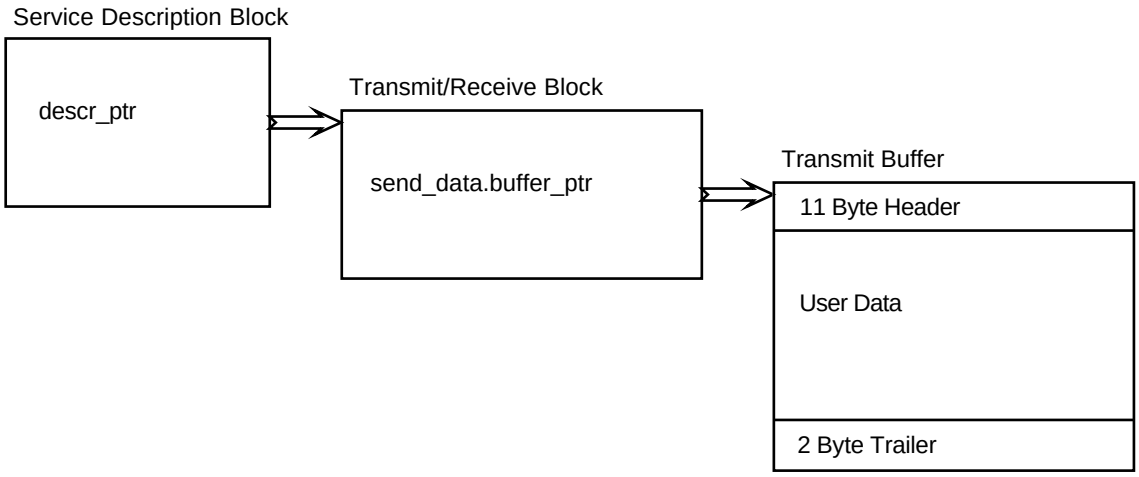
 © 1995 PEP Modular Computers

SDA (Send Data with Acknowledge) Request

Description:

The local station sends data to a remote station (via *rem_add*) and awaits confirmation of a valid or errornous transfer.

Data Structure:



The FDL header and trailer in the transmit buffer are automatically created by the FDL and sufficient memory for these additions must be made available by the FDL-User. The data may be written to the transmit buffer starting at the 12th byte.

The three linked structures remain in the layer 2 until confirmation of successful or errornous transfer is returned by the target station. Therefore the allocated memory cannot be used for anything else until this has been completed.

Service Description Block:

sap	0..62 or DEFAULT_SAP	Local Service Access Point
service	SDA	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_SR_BLOCK far*)	Pointer to transmit/receive block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Transmit/Receive Block:

loc_add.station	*not significant*	
loc_add.segment	*not significant*	
remote_sap	0..63 or DEFAULT_SAP	Destination Service Access Point
rem_add.station	0..126	Remote station address
rem_add.segment	0..63 or NO_SEGMENT	Remote segment address
serv_class	LOW or HIGH	Priority of the service call
update_status	*not significant*	
send_data.buffer_ptr	(UNSIGN8 far*)	Pointer to transmit buffer
send_data.length	1..242	Length of user data
receive_data.buffer_ptr	*not significant*	
receive_data.length	*not significant*	
resource.buffer_ptr	*not significant*	
resource.length	*not significant*	

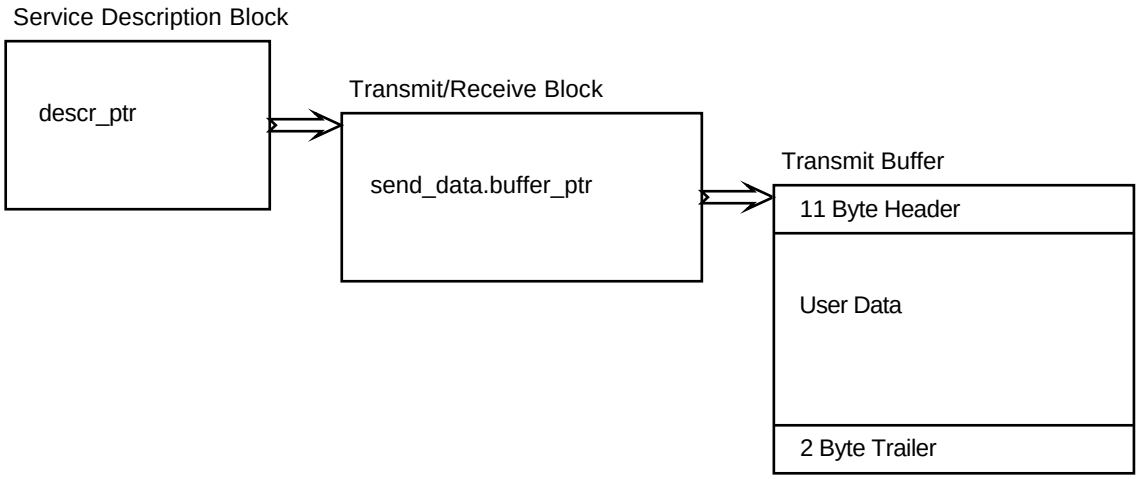
 © 1995 PEP Modular Computers

SDA (Send Data with Acknowledge) Confirmation

Description:

The remote station sends confirmation of a valid or errornous completion of an SDA request to the FDL-User (message originator). If the confirmation cannot be sent due to local circumstances a negative status automatically occurs.

Data Structure:



The data structure passed by the FDL request is returned to the FDL-User. The positive or negative confirmation of successful transfer is given in the status field.

Service Description Block:

sap	*remains unchanged*	
service	SDA	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, RR, RS, LS, NA, IV, NLT	*see below*
descr_ptr	*remains unchanged*	Pointer to transmit/receive block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Transmit/Receive Block:

loc_add.station	*not significant*	
loc_add.segment	*not significant*	
remote_sap	*remains unchanged*	
rem_add.station	*remains unchanged*	
rem_add.segment	*remains unchanged*	
serv_class	*remains unchanged*	
update_status	*not significant*	
send_data.buffer_ptr	*remains unchanged*	Pointer to transmit buffer
send_data.length	*remains unchanged*	
receive_data.buffer_ptr	*not significant*	
receive_data.length	*not significant*	
resource.buffer_ptr	*not significant*	
resource.length	*not significant*	

 © 1995 PEP Modular Computers

Status Values:

Code	Meaning
OK	Positive confirmation that the service has been carried out
RR	The partner did not have adequate operational resources
RS	Partners service, access authorization or SAP, is not activated
LS	Service or local Service Access Point not activated
NA	Addressed partner does not respond
IV	Invalid parameter in request
NLT	Own station not in logical token ring

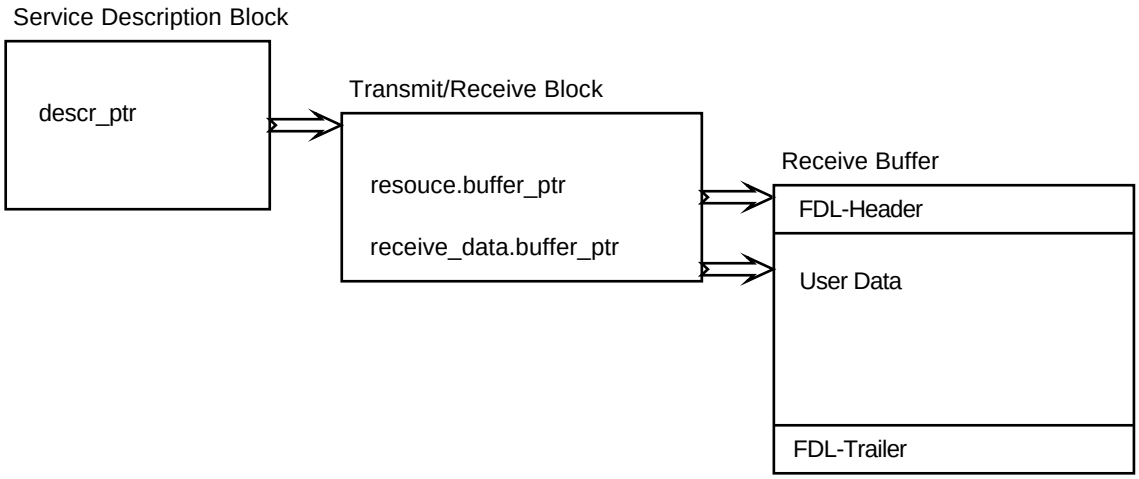
© 1995 PEP Modular Computers

SDA (Send Data with Acknowledge) Indication

Description:

The FDL indicates to the local station that data has been received as a result of an SDA request service initiated by a remote service.

Data Structure:



The *receive_data.buffer_ptr* is used to point to the start of the valid user data. *resource.buffer_ptr* points to the start of the buffer i.e. the resource transferred to the FDL via *PUT_RESCRC_TO_FDL*.

Service Description Block:

sap	0..63 or DEFAULT_SAP	Destination Service Access Point (DSAP)
service	SDA	
primitive	IND	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_SR_BLOCK far*)	Pointer to transmit/receive block
next_descr	reserved for FDL	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Transmit/Receive Block:

loc_add.station	0..126	Source station address
loc_add.segment	0..63 or NO_SEGMENT	Source segment address
remote_sap	0..62 or DEFAULT_SAP	Source Service Access Point (SSAP)
rem_add.station	0..126	Remote station address
rem_add.segment	0..63 or NO_SEGMENT	Remote segment address
serv_class	LOW or HIGH	Priority of the service call
update_status	*not significant*	
send_data.buffer_ptr	*not significant*	
send_data.length	*not significant*	
receive_data.buffer_ptr	(UNSIGN8 far*)	Pointer to user data
receive_data.length	1..242	Length of received user data
resource.buffer_ptr	(UNSIGN8 far*)	Pointer to receive buffer
resource.length	<= 255	Length of receive buffer

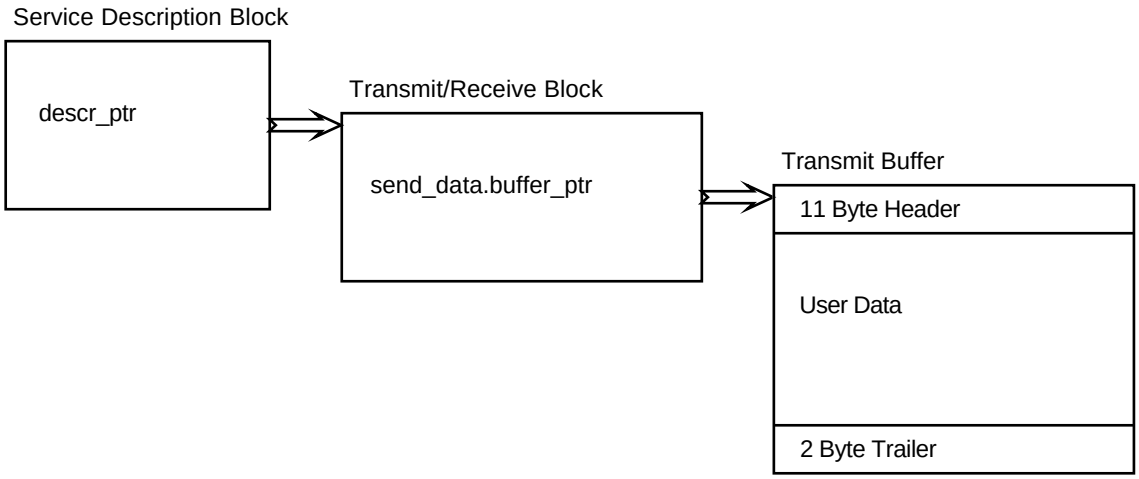
 © 1995 PEP Modular Computers

SDN (Send Data with No Acknowledge) Request

Description:

The local station sends data to a group or all remote stations. The service is not confirmed by the recipients, but rather a local "sent"receipt is generated.

Data Structure:



The FDL header and trailer in the transmit buffer are automatically created by the FDL and sufficient memory for these additions must be made available by the FDL-User. The data may be written to the transmit buffer starting at the 12th byte.

The three linked structures remain in the layer 2 until confirmation. Therefore the allocated memory cannot be used for anything else until this has been successfully completed.

Service Description Block:

sap	0..62 or DEFAULT_SAP	Local Service Access Point
service	SDN	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_SR_BLOCK far*)	Pointer to transmit/receive block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Transmit/Receive Block:

loc_add.station	*not significant*	
loc_add.segment	*not significant*	
remote_sap	0..63 or DEFAULT_SAP	Destination Service Access Point
rem_add.station	0..126 or global address 127	Remote station address(es)
rem_add.segment	0..63 or NO_SEGMENT	Remote segment address
serv_class	LOW or HIGH	Priority of the service call
update_status	*not significant*	
send_data.buffer_ptr	(UNSIGN8 far*)	Pointer to transmit buffer
send_data.length	1..242	Length of user data
receive_data.buffer_ptr	*not significant*	
receive_data.length	*not significant*	
resource.buffer_ptr	*not significant*	
resource.length	*not significant*	

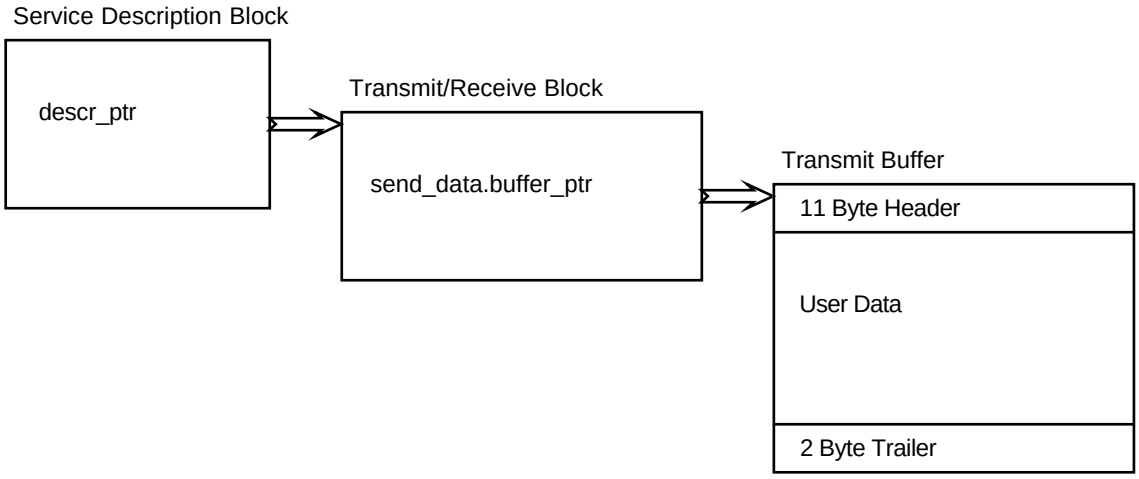
 © 1995 PEP Modular Computers

SDN (Send Data with No Acknowledge) Confirmation

Description:

The local station's FDL generates confirmation if no errornous transfer messages are returned.

Data Structure:



The data structures passed by the FDL request are returned to the FDL-User. The positive or negative confirmation is shown in the status field.

Service Description Block:

sap	*remains unchanged*	
service	SDN	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, LS, IV, NLT	
descr_ptr	*remains unchanged*	Pointer to transmit/receive block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Transmit/Receive Block:

loc_add.station	*not significant*	
loc_add.segment	*not significant*	
remote_sap	*remains unchanged*	
rem_add.station	*remains unchanged*	
rem_add.segment	*remains unchanged*	
serv_class	*remains unchanged*	
update_status	*not significant*	
send_data.buffer_ptr	*remains unchanged*	Pointer to transmit buffer
send_data.length	*remains unchanged*	Length of user data
receive_data.buffer_ptr	*not significant*	
receive_data.length	*not significant*	
resource.buffer_ptr	*not significant*	
resource.length	*not significant*	

 © 1995 PEP Modular Computers

Status Values:

Code	Meaning
OK	Positive confirmation that the service has been carried out
LS	Service or local Service Access Point not activated
IV	Invalid parameter in request
NLT	Own station not in logical token ring

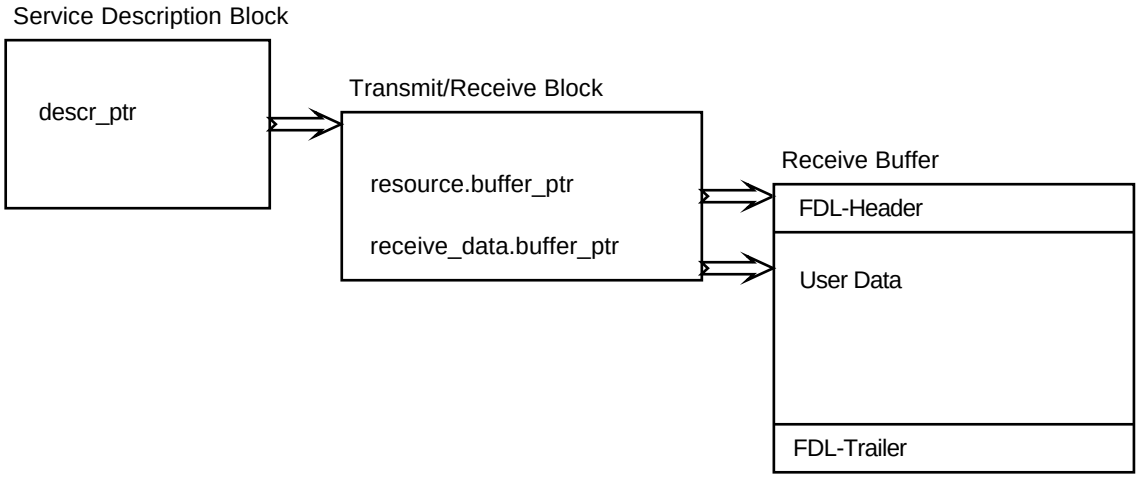
 © 1995 PEP Modular Computers

SDN (Send Data with No Acknowledge) Indication

Description:

The FDL indicates that the local station has received data via an SDN request service initiated by a remote station.

Data Structure:



The *receive_data.buffer_ptr* is used to point to the start of the valid user data. *resource.buffer_ptr* points to the start of the buffer i.e. the resource to be transferred to the FDL via *PUT_RESCRC_TO_FDL*.

Service Description Block:

sap	0..63 or DEFAULT_SAP	Destination Service Access Point (DSAP)
service	SDN	
primitive	IND	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_SR_BLOCK far*)	Pointer to transmit/receive block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Transmit/Receive Block:

loc_add.station	0..126	Source station address
loc_add.segment	0..63 or NO_SEGMENT	Source segment address
remote_sap	0..62 or DEFAULT_SAP	Source's Service Access Point (SSAP)
rem_add.station	0..126 or global address 127	Remote station address
rem_add.segment	0..63 or NO_SEGMENT	Remote segment address
serv_class	LOW or HIGH	Priority of the service call
update_status	*not significant*	
send_data.buffer_ptr	*not significant*	
send_data.length	*not significant*	
receive_data.buffer_ptr	(UNSIGN8 far*)	Pointer to user data
receive_data.length	1..242	Length of the received user data
resource.buffer_ptr	(UNSIGN8 far*)	Pointer to receive buffer
resource.length	<= 255	Length of receive buffer

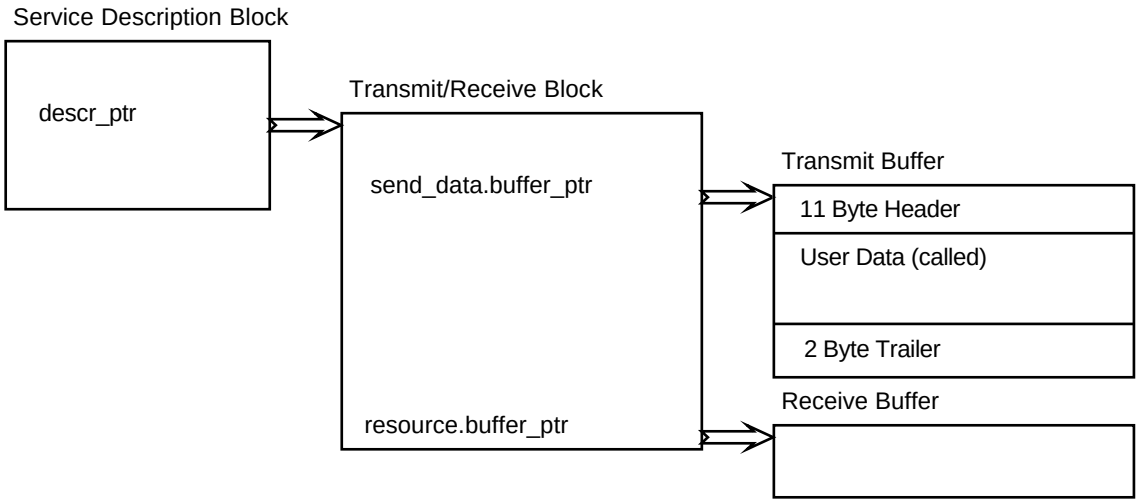
 © 1995 PEP Modular Computers

SRD (Send and Request Data with Reply) Request

Description:

The local station sends data to defined station(s) (via *rem_add*) and collects any data waiting there. If no data is present the local station only receives a receipt.

Data Structure:



The FDL header and trailer in the transmit buffer are automatically created by the FDL and sufficient memory for these additions must be made available by the FDL-User. The data may be written to the transmit buffer starting at the 12th byte.

A receive buffer must be provided by the FDL-User via *resource.buffer_ptr* for any replied data.

Service Description Block:

sap	0..62 or DEFAULT_SAP	Local Service Access Point
service	SRD	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_SR_BLOCK far*)	Pointer to transmit/receive block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Transmit/Receive Block:

loc_add.station	*not significant*	
loc_add.segment	*not significant*	
remote_sap	0..62 or DEFAULT_SAP	Destination Service Access Point
rem_add.station	0..126	Remote station address
rem_add.segment	0..63 or NO_SEGMENT	Remote segment address
serv_class	LOW or HIGH	Priority of the service call
update_status	*not significant*	
send_data.buffer_ptr	(UNSIGN8 far*)	Pointer to transmit buffer
send_data.length	0..242	Length of user data
receive_data.buffer_ptr	*not significant*	
receive_data.length	*not significant*	
resource.buffer_ptr	(UNSIGN8 far*)	Pointer to reply buffer
resource.length	<= 255	Length of reply buffer

Note: The data structures remain in level 2 and allocated memory cannot be used until completion of their task.

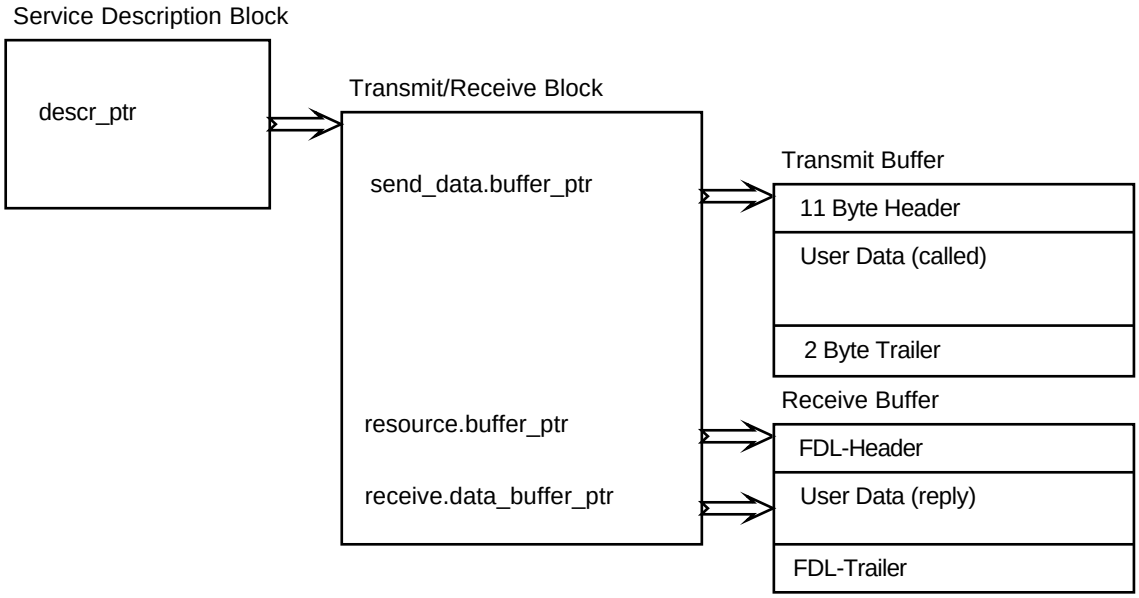
 © 1995 PEP Modular Computers

SRD (Send and Request Data with Reply) Confirmation

Description:

The confirmation of a valid or erranous completion of the SRD request returned and also indicates if any reply data is available.

Data Structure:



The data structures passed by the FDL request are returned to the FDL-User.

Service Description Block:

sap	*remains unchanged*	
service	SRD	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	RS, LS, LR, NA, IV, DL, DH, NR, RDL, RDH, RR, NLT	
descr_ptr	*remains unchanged*	Pointer to transmit/receive block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Transmit/Receive Block:

loc_add.station	*not significant*	
loc_add.segment	*not significant*	
remote_sap	*remains unchanged*	
rem_add.station	*remains unchanged*	
rem_add.segment	*remains unchanged*	
serv_class	*remains unchanged*	
update_status	*not significant*	
send_data.buffer_ptr	*remains unchanged*	Pointer to transmit buffer
send_data.length	*remains unchanged*	
receive_data.buffer_ptr	(USIGN8 far *)	Pointer to reply data buffer
receive_data.length	0..242	Length of reply data
resource.buffer_ptr	*remains unchanged*	Pointer to reply telegram
resource.length	*remains unchanged*	Length of reply telegram

Status Values:

Code	Meaning
RS	Partners service, access authorization or SAP, is not activated
LS	Service or local Service Access Point not activated
LR	None or insufficient operational resources are available locally
NA	Addressed partner does not respond
IV	Invalid parameter in request
DL	Reply data low available, positive conformation of data sent
DH	Reply data high available, positive conformation of data sent
NR	No reply data available, positive conformation of data sent
RDL	Reply data low available, negative conformation of data sent
RDH	Reply data high available, negative conformation of data sent
RR	The partner did not have adequate operational resources
NLT	Own station not in logical token ring

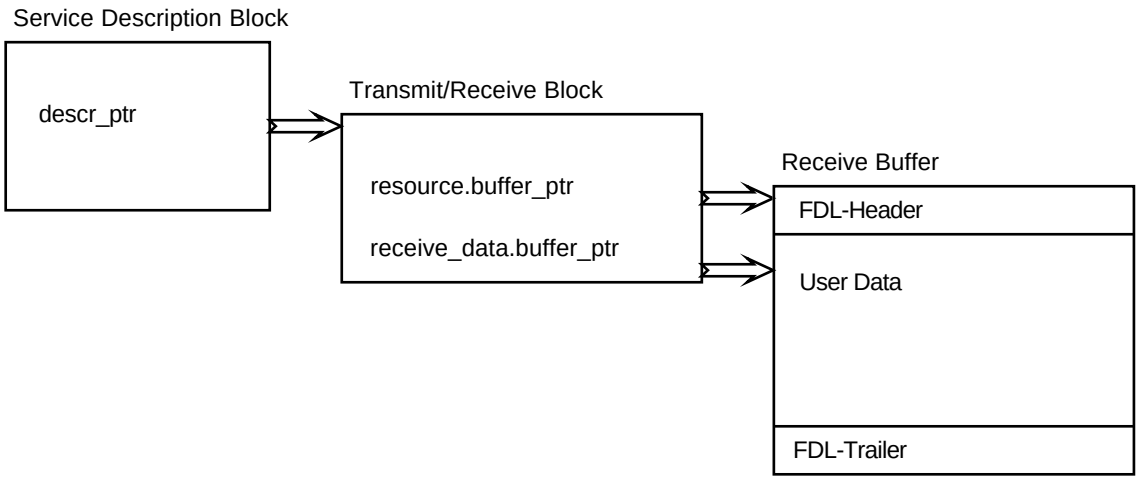
 © 1995 PEP Modular Computers

SRD (Send and Request Data with Reply) Indication

Description:

The FDL indicates that another station has completed an SRD cycle with the local station. If the Responder-SAP is defined as 'indication_mode==DATA' (see service *FMA2_ACTIVATE_RSAP*) only SRD cycles where data has been transferred, either with the receipt or the reply telegram, are indicated. If the Responder-SAP is defined as 'indication_mode== ALL', the SRD cycles without data are also indicated.

Data Structure:



The *receive_data.buffer_ptr* is used to point to the start of the valid user data. *resource.buffer_ptr* points to the start of the buffer i.e. the resource to be transferred to the FDL via *PUT_RESCRC_TO_FDL*.

Service Description Block:

sap	0..62 or DEFAULT_SAP	Destination Service Access Point (DSAP)
service	SRD	
primitive	IND	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_SR_BLOCK far *)	Pointer to transmit/receive block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Transmit/Receive Block:

loc_add.station	0..126	Source station address
loc_add.segment	0..63 or NO_SEGMENT	Source segment address
remote_sap	0..62 or DEFAULT_SAP	Source Service Access Point (SSAP)
rem_add.station	0..126	Remote station address
rem_add.segment	0..63 or NO_SEGMENT	Remote segment address
serv_class	LOW or HIGH	Priority of the service call
update_status	NO, LO, HI	Status of the reply data sent
send_data.buffer_ptr	*not significant*	
send_data.length	*not significant*	
receive_data.buffer_ptr	(UNSIGN8 far *)	Pointer to user data
receive_data.length	0..242	Length of the received user data
resource.buffer_ptr	(UNSIGN8 far *)	Pointer to receiver buffer
resource.length	<= 255	Length of receiver buffer

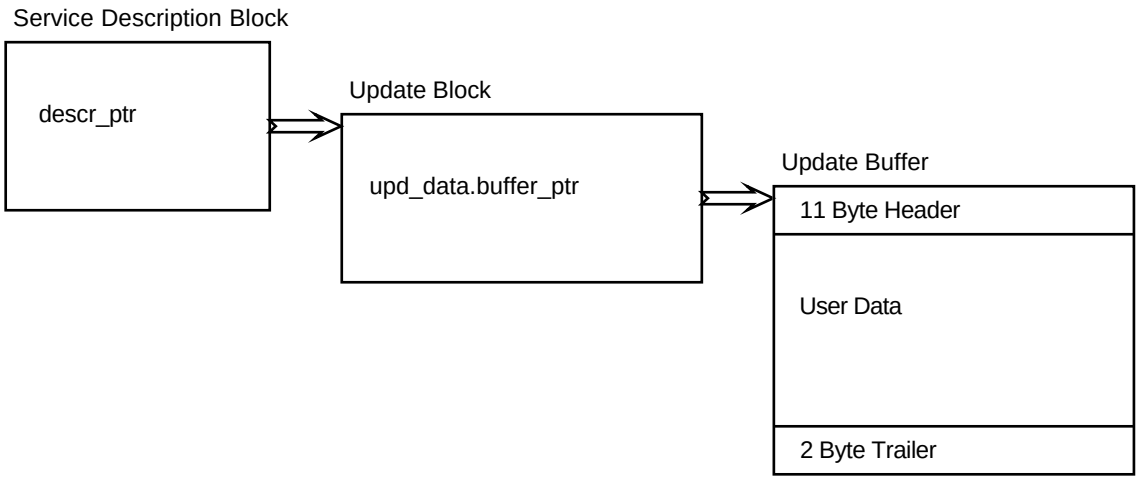
 © 1995 PEP Modular Computers

REPLY_UPDATE Request

Description:

This primitive is used by the FDL-User to transfer data to a Service Access Point that was activated by the service *FMA2_ACTIVATE_RSAP*. The data can be collected by another participant with either an SRD or CSRD service call. This transfer can be either singular (transmit = SINGLE) or multiple (transmit = MULTIPLE) as desired. The confirmation of transfer occurs with the next SRD.ind.

Data Structure:



The FDL header and trailer in the update buffer are automatically created by the FDL and sufficient memory for these additions must be made available by the FDL-User. The data may be written to the transmit buffer starting at the 12th byte.

The Service Description Block and the update block remain in the FDL until the respective confirmation of successful or errornous transfer is completed. The update buffer is returned only with the confirmation of the next update call and must remain available for the FDL until the next update request.

Service Description Block:

sap	0..62 or DEFAULT_SAP	Local Service Access Point
service	REPLY__UPDATE	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_UPDATE_BLOCK far *)	Pointer to update block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Update Block:

dsap	0..62 or DEFAULT_SAP	Destination Service Access Point
rem_add.station	0..126	Remote station address
rem_add.segment	0..63 or NO_SEGMENT	Remote segment address
serv_class	LOW or HIGH	Priority of the service call
transmit	SINGLE or MULTIPLE	
upd_data.buffer_ptr	(UNSIGN8 far *)	Pointer to update buffer
upd_data.length	1..242	Length of user data

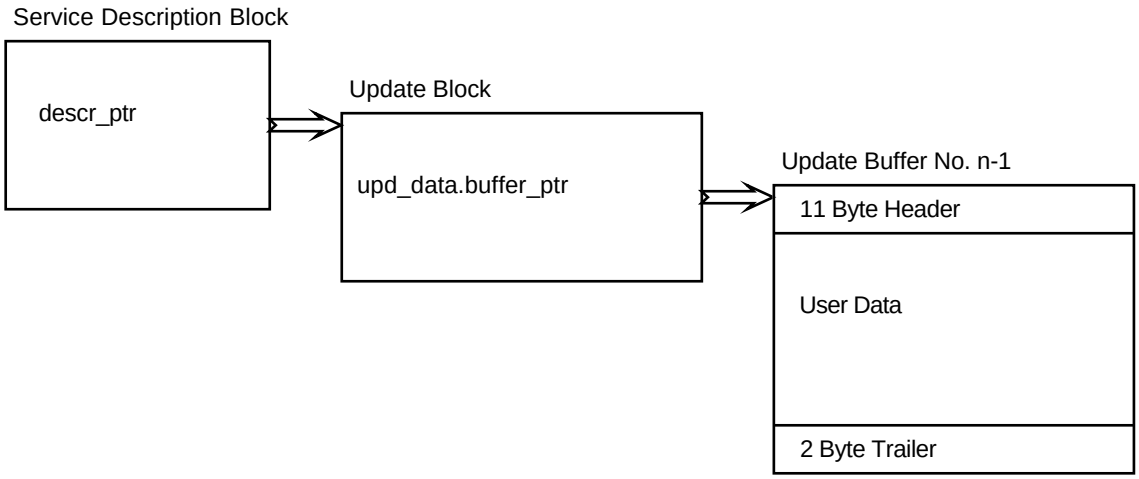
 © 1995 PEP Modular Computers

REPLY_UPDATE Confirmation

Description:

The FDL confirms acceptance of the update buffer. If the confirmation is positive, the update buffer remains in the FDL until the following UPDATE.req, returning no buffer for the first UPDATE.con. Hence UPDATE.con *n* always returns the update buffer of the UPDATE.req *n-1*.
The confirmation of a successful transfer of the update buffer contents is indicated with a SRD.ind.

Data Structure:



If the status is OK, the parameter *upd_data.buffer_ptr* returns the value NULL (zero) for the first confirmation. Subsequent confirmations return the update buffer, which was sent to the FDL with the preceding UPDATE.req (n-1).

Service Description Block:

sap	*remains unchanged*	
service	REPLY_UPDATE	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, LS, LR, IV	*see below*
descr_ptr	*remains unchanged*	Pointer to update block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Update Block:

dsap	*remains unchanged*	
rem_add.station	*remains unchanged*	
rem_add.segment	*remains unchanged*	
serv_class	*remains unchanged*	
transmit	*remains unchanged*	
upd_data.buffer_ptr	(UNSIGN8 far *)	Pointer to update buffer of call *n-1*
upd_data.length	1..242	

 © 1995 PEP Modular Computers

Status Values:

Code	Meaning
OK	Update buffer has been accepted
LS	Service Access Point not activated
LR	During SINGLE, no transfer since previous update buffer has not been sent
IV	Invalid parameter in request

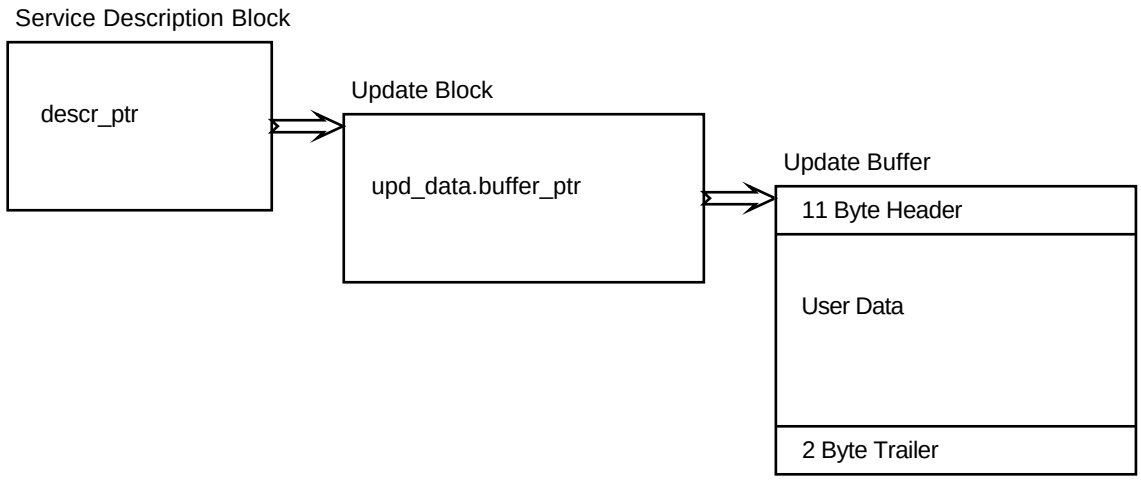
 © 1995 PEP Modular Computers

SEND_UPDATE Request

Description:

With this service the FDL-User can pass data to the Poll-List 'administration' of the FDL. The Poll-List entry is determined by the *rem_add* and *dsap* combination. The data is transferred using an SRD cycle when the respective Poll-List entry is handled, either as a single (transmit = SINGLE) or as a multiple (transmit = MULTIPLE) transfer. Confirmation of the transfer occurs with a CSRD.con.

Data Structure:



The FDL header and trailer in the update buffer are automatically created by the FDL and sufficient memory for these additions must be made available by the FDL-User. The data may be written to the transmit buffer starting at the 12th byte.

The Service Description Block and the update block remain in the FDL until the respective confirmation of successful or errornous transfer is completed. The update buffer is returned only with the confirmation of the next update call and must remain available for the FDL until the next update request.

Service Description Block:

sap	0..62 or DEFAULT_SAP	Service Access Point of Poll-List (Poll-SAP)
service	SEND__UPDATE	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_UPDATE_BLOCK far *)	Pointer to update block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Update Block:

dsap	0..62 or DEFAULT_SAP	
rem_add.station	0..126	Remote station address
rem_add.segment	0..63 or NO_SEGMENT	Remote segment address
serv_class	LOW or HIGH	Priority of the service call
transmit	SINGLE or MULTIPLE	
upd_data.buffer_ptr	(UNSIGN8 far *)	Pointer to update buffer
upd_data.length	1..242	Length of user data

 © 1995 PEP Modular Computers

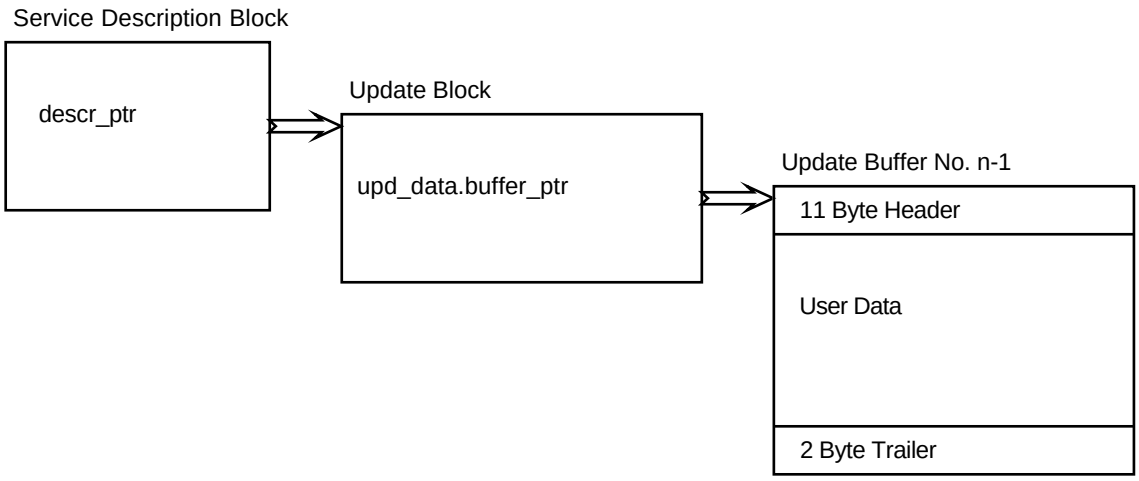
SEND_UPDATE Confirmation

Description:

The FDL confirms acceptance of the update buffer into the Poll-List designated with the *rem_add*/*dsap* combination. If the confirmation is positive, the update buffer remains in the FDL until the following UPDATE.req, returning no buffer for the first UPDATE.con. Hence UPDATE.con *n* always returns the update buffer of the UPDATE.req *n-1*.

The confirmation of a successful transfer of the update buffer contents is indicated with a CSRD.con.

Data Structure:



If the status is OK, the parameter "upd_data.buffer_ptr" returns the value NULL (zero) for the first confirmation. Subsequent confirmations return the update buffer, which was sent to the FDL with the preceding UPDATE.req (n-1).

Service Description Block:

sap	*remains unchanged*	
service	SEND_UPDATE	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, LS, LR, IV	*see below*
descr_ptr	*remains unchanged*	Pointer to update block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Update Block:

dsap	*remains unchanged*	
rem_add.station	*remains unchanged*	
rem_add.segment	*remains unchanged*	
serv_class	*remains unchanged*	
transmit	*remains unchanged*	
upd_data.buffer_ptr	(UNSIGN8 far *)	Pointer to update buffer of call *n-1*
upd_data.length	1..242	

 © 1995 PEP Modular Computers

Status Values:

Code	Meaning
OK	Update buffer has been accepted
LS	Service Access Point not activated
LR	*rem_add*/*dsap* combination not found in the Poll-List. During SINGLE, no transfer since previous update buffer has not been sent
IV	Invalid parameter in request

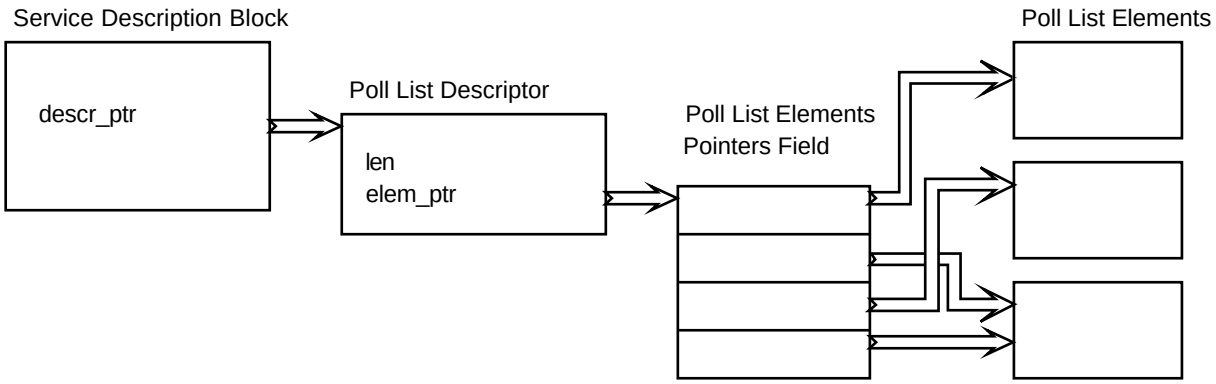
 © 1995 PEP Modular Computers

LOAD_POLL_LIST Request

Description:

With this service the Poll-List is given to the FDL-User.

Data Structure:



The Service Description Block contains a pointer to the Poll-List descriptor, which contain details about the number and location of the pointer fields. The length of the pointer field must exactly represent the number of Poll-List entries required. Every entry must point to a Poll-List element which in turn will contain administrative information required for the transfer.

The pointer field represents the Poll-List according to the DIN 19245 standard. This is because the priority of the Poll-List elements can be manipulated by multiple entries in the pointer field to the same Poll-List element and the arrangement of the pointer in the pointer field allows a determination of the execution sequence.

Service Description Block:

sap	0..62	Service Access Point of Poll-List (Poll-SAP)
service	LOAD_POLL_LIST	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_POLL_LIST_DESCR far *)	Pointer to Poll-List descriptor
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Poll List Descriptor:

len	>0	Number of Poll-List entries
confirm_mode	ALL or DATA	ALL: all SRD cycles are confirmed by CSRD-confirmation DATA: Only SRD cycles with user data are confirmed
elem_ptr	(T_POLL_LIST_ELEM_PTR far *)	Pointer to pointer field

Pointer Field:

T_POLL_LIST_ELEM_PTR [quantity of Poll-List entries]

 © 1995 PEP Modular Computers

Poll-List Elements:

dsap	0..62 or DEFAULT_SAP	Destination Service Access Point
rem_add.station	0..126	
rem_add.segment	0..63 or NO_SEGMENT	
max_len_csrd_req_low	0..242	Maximum length of request (low)
max_len_csrd_con_low	0..242	Maximum length of reply data (low)
max_len_csrd_con_high	0..242	Maximum length of reply data (high)
poll_buffer.buffer_ptr	(UNSIGN8 far *)	Pointer to buffer for poll telegrams
poll_buffer.length	> = FDL_OFFSET+FDL_TRAILER	Length of buffer
send_data	*not significant*	
resrc_hdr	*not significant*	
resrc_tail	*not significant*	
resrc_ctr	*not significant*	
transmit	*reserved for FDL*	
to_send	*reserved for FDL*	
marker	*reserved for FDL*	
poll_telegram	*reserved for FDL*	
data_telegram	*reserved for FDL*	
data_fcs	*reserved for FDL*	
poll_fcs	*reserved for FDL*	

Note: Multiple entries in the Poll-List are achieved by including more than one entry in the Pointers Field to the same Poll-List element.

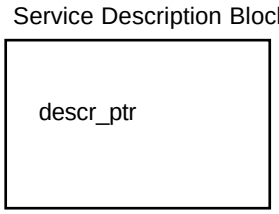
 © 1995 PEP Modular Computers

LOAD_POLL_LIST Confirmation

Description:

The FDL confirms acceptance of the Poll-List or rejects the Poll-List with an error status.

Data Structure:



If the status is OK, only the Service Description Block is returned, otherwise the entire structure transfered with the request is returned indicating an error.

Service Description Block:

sap	0..62	Service Access Point for Poll-List (Poll-SAP)
service	LOAD_POLL_LIST	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, NO, IV	*see below*
descr_ptr	NULL (= zero)	By error remains unchanged
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Note: Value of *descr_ptr*: When status = OK this counter is reset to 0, otherwise it points to the Poll-List descriptor given with the request.

Status Values:

Code	Meaning
OK	The Poll-List has been accepted by the FDL
NO	The FDL already contains a Poll-List
IV	Invalid parameter in request

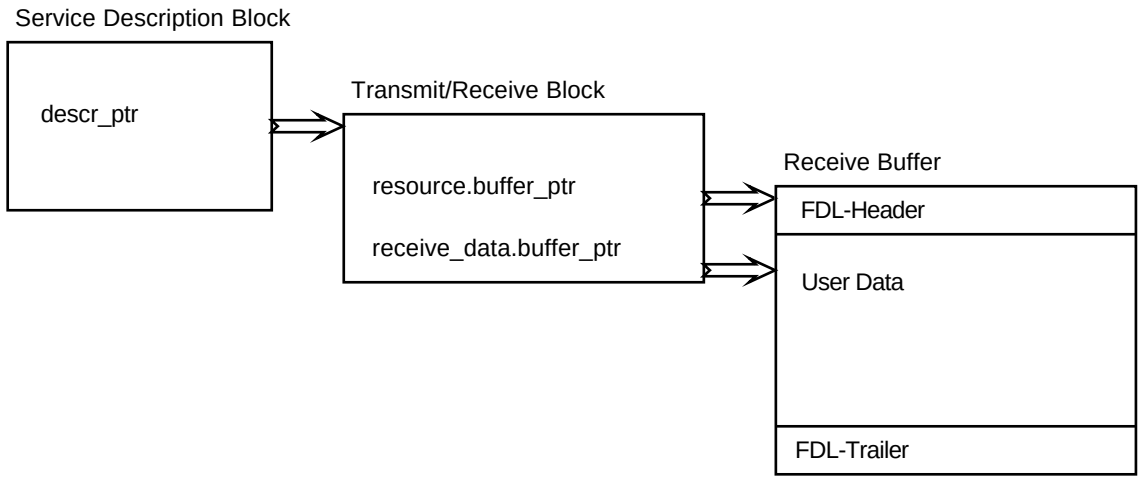
 © 1995 PEP Modular Computers

Cyclic Send and Request Data with Reply (CSRD) Confirmation

Description:

This primitive indicates to the FDL that an SRD cycle is being processed with the Poll-List and the status is returned for any eventually received reply data. Subject to the value of the "confirm_mode" parameter as loaded into the Poll-List descriptor, all SRD cycles (confirm_mode = ALL) are indicated to the FDL-User or only the SRC cycles are indicated that contain user data either in the sent or in the receipt telegram or in both (confirm_mode = DATA).

Data Structure:



The *receive_data.buffer_ptr* is used to point to the start of the valid user data. *resource.buffer_ptr* points to the start of the buffer i.e. the resource to be transferred to the FDL via *PUT_RESCRC_TO_FDL*.

Service Description Block:

sap	0..62 or DEFAULT_SAP	Service Access Point (SAP) of Poll-List
service	CSRD	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	RS, LS, LR, NA, IV, NLT	
	DL, DH, NR, RDL, RDH, RR	*see below*
descr_ptr	(T_FDL_SR_BLOCK far *)	Pointer to transmit/receive block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Transmit/Receive Block:

loc_add.station	0..126	Local station address
loc_add.segment	0..63 or NO_SEGMENT	Local segment address
remote_sap	0..62 or DEFAULT_SAP	Partner's Service Access Point (SAP)
rem_add.station	0..126	Remote station address
rem_add.segment	0..63 or NO_SEGMENT	Remote segment address
serv_class	LOW or HIGH	Priority of the service call
update_status	NO, LO	Status of the calling data sent
send_data.buffer_ptr	*not significant*	
send_data.length	*not significant*	
receive_data.buffer_ptr	(UNSIGN8 far *)	Pointer to user data
receive_data.length	1..242	Length of the received user data
resource.buffer_ptr	(UNSIGN8 far *)	Pointer to receiver buffer
resource.length	<= 255	Length of receiver buffer

Status Values:

Code	Meaning
RS	Partners service, access authorization or SAP, is not activated
LS	Service or local Service Access Point not activated
LR	None or insufficient operational resources are available locally
NA	Addressed partner does not respond
IV	Invalid parameter in request
DL	Reply data low available, positive conformation of data sent
DH	Reply data high available, positive conformation of data sent
NR	No reply data available, positive conformation of data sent
RDL	Reply data low available, negative conformation of data sent
RDH	Reply data high available, negative conformation of data sent
RR	The partner did not have adequate operational resources
NLT	Own station not in logical token ring

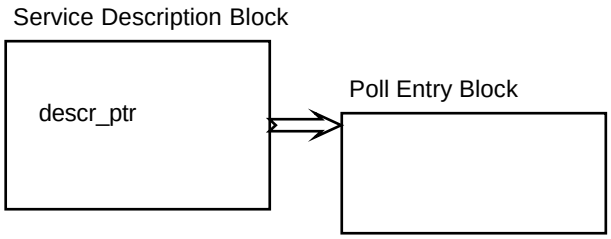
 © 1995 PEP Modular Computers

POLL_ENTRY Request

Description:

Through the given *rem_add*/*dsap* combination, a specific entry in the Poll-List is marked as either available or barred by the value of the "marker" parameter of the poll entry block. Thus it is possible to temporarily deactivate an entry in the Poll-List, and save the partner having to poll it along with the currently desired entries.

Data Structure:



Service Description Block:

sap	0..62 or DEFAULT_SAP	Service Access Point of Poll-List (Poll-SAP)
service	POLL_ENTRY	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_POLL_ENTRY far *)	Pointer to poll entry block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Poll Entry Block:

dsap	0..62 or DEFAULT_SAP	Service Access Point of the partner
rem_add.station	0..126	Station address of the partner
rem_add.segment	0..63 or NO_SEGMENT	Segment address of the partner
marker	LOCKED or UNLOCKED	New state of the entry

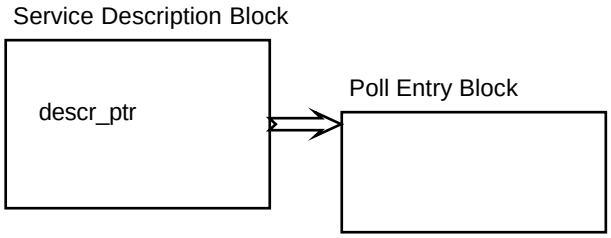
 © 1995 PEP Modular Computers

POLL_ENTRY Confirmation

Description:

The availability or barred state of a Poll-List entry is confirmed.

Data Structure:



Service Description Block:

sap	*remains unchanged*	Service Access Point of Poll-List (Poll-SAP)
service	POLL_ENTRY	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, LS, NO, IV	*see below*
descr_ptr	(T_POLL_ENTRY far *)	Pointer to poll entry block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Poll Entry Blocks:

dsap	*remains unchanged*	Partner's Service Access Point
rem_add.station	*remains unchanged*	Station address of the partner
rem_add.segment	*remains unchanged*	Segment address of the partner
marker	*remains unchanged*	New state of the entry

Status Values:

Code	Meaning
OK	Marker set
LS	No Poll-List exists in the FDL at this Service Access Point
NO	Marker not set, *rem_add*/*dsap* combination not found in the Poll-List
IV	Invalid parameter in request

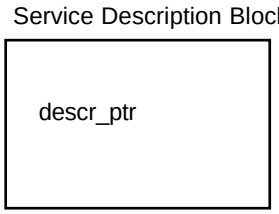
 © 1995 PEP Modular Computers

DEACT_POLL_LIST Request

Description:

The processing of the Poll-List is terminated after completion of the current activity.

Data Structure:



It only remains to complete the Service Description Block.

Service Description Block:

sap	0..62 or DEFAULT_SAP	Service Access Point for Poll-List (Poll-SAP)
service	DEACT_POLL_LIST	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	*not significant*	
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

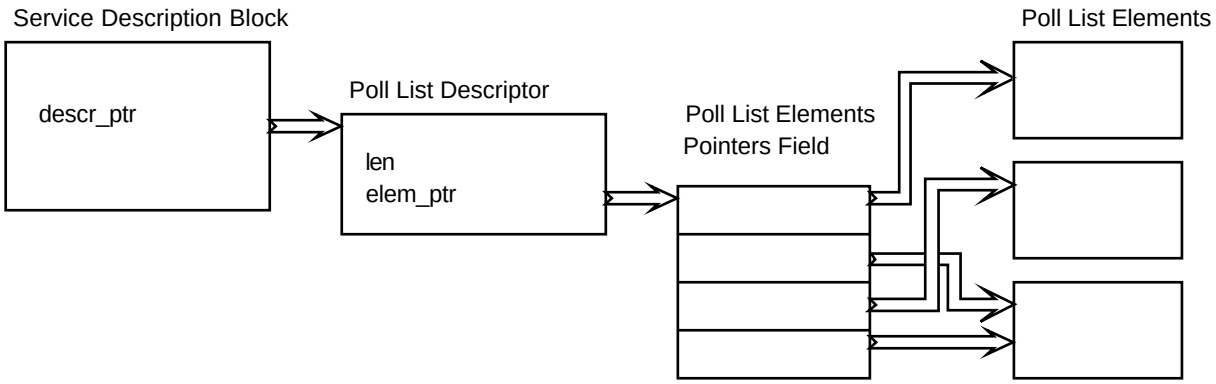
 © 1995 PEP Modular Computers

DEACT_POLL_LIST Confirmation

Description:

The deactivation of the Poll-List is confirmed and if status = "OK" the Poll-List is given back to the FDL-User.

Data Structure:



The structure of the Poll-List is identical to the structure given over during the *LOAD_POLL_LIST*.request. In the Poll-List elements under the *resc_ctr*, the number of resources allocated to the respective elements are given.

The resource *resc_tail* points to the linked resources, and the *send_data* contains the last update buffer.

Service Description Block:

sap	*remains unchanged*	Service Access Point of Poll-List (Poll-SAP)
service	DEACT_POLL_LIST	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, LS or IV	*see below*
descr_ptr	(T_POLL_LIST_DESCR far *)	Pointer to Poll-List descriptor
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Status Values:

Code	Meaning
OK	Poll-List is disabled
LS	The specified Service Access Point does not have a Poll-List in the FDL
IV	Invalid parameter in request

 © 1995 PEP Modular Computers

4.5.7 FMA Services

The FMA (Fieldbus Management) services are made available through the management (FMA1/2) associated with the layers 1 and 2. A list of the FMA1/2 services as defined in DIN 19245, Part 1 follows below. On the right hand side two columns form constants "*service*" and "*primitive*" as they must be given in the Service Description Block (*T_FDL_SERVICE_DESCR*). These terms are agreed for use in the include-data "`pbL2con.h`".

The services *FMA1/2_SET_VALUE* and *FMA1/2_READ_VALUE* are created by multiple intercommunication services, due to their many varied parameter structures.

Terminology to DIN 19245, Part 1 Intercommunication Interface

	Service	Primitive	Possible for
Reset FMA1/2			
FMA1/2_RESET.request	*FMA2_RESET*	*REQ*	*M and S*
FMA1/2_RESET.confirm	*FMA2_RESET*	*CON*	*M and S*
Set Value FMA1/2			
FMA1/2_SET_VALUE.request			
FMA1/2_SET_VALUE.confirm			
	FMA2_SET_BUSPARAMETER	*REQ*	*M and S*
	FMA2_SET_BUSPARAMETER	*CON*	*M and S*
	FMA2_SET_STATISTIC_CTR	*REQ*	*M and S*
	FMA2_SET_STATISTIC_CTR	*CON*	*M and S*
	FMA2_CHANGE_BUSPARAMETER	*REQ*	*M and S*
	FMA2_CHANGE_BUSPARAMETER	*CON*	*M and S*
Read Value FMA1/2			
FMA1/2_READ_VALUE.request			
FMA1/2_READ_VALUE.confirm			
	FMA2_READ_BUSPARAMETER	*REQ*	*M and S*
	FMA2_READ_BUSPARAMETER	*CON*	*M and S*
	FMA2_READ_STATISTIC_CTR	*REQ*	*M and S*
	FMA2_READ_STATISTIC_CTR	*CON*	*M and S*
	FMA2_READ_TRR	*REQ*	*M*
	FMA2_READ_TRR	*CON*	*M*
	FMA2_READ_LAS	*REQ*	*M*
	FMA2_READ_LAS	*CON*	*M*
	FMA2_READ_GAPLIST	*REQ*	*M*
	FMA2_READ_GAPLIST	*CON*	*M*

 © 1995 PEP Modular Computers

Event FMA1/2

FMA1/2_EVENT.indication	*FMA2_EVENT*	*IND*	*M and S*

Ident FMA1/2

FMA1/2_IDENT.request	*FMA2_LSAP_STATUS*	*REQ*	*local: M and S* *remote : M*
FMA1/2_IDENT.confirm	*FMA2_LSAP_STATUS*	*CON*	*local: M and S* *remote: M*

LSAP FMA1/2

FMA1/2_LSAP_STATUS.request	*FMA2_IDENT*	*REQ*	*local: M and S* *remote: M*
FMA1/2_LSAP_STATUS.confirm	*FMA2_IDENT*	*CON*	*local:M and S* *remote: M*

Live-List FMA1/2

FMA1/2_LIVE_LIST.request	*FMA2_LIVELIST*	*REQ*	*M*
FMA1/2_LIVE_LIST.confirm	*FMA2_LIVELIST*	*CON*	*M*

SAP Activate FMA1/2

FMA1/2_SAP_ACTIVATE.request	*FMA2_ACTIVATE_SAP*	*REQ*	*M and S*
FMA1/2_SAP_ACTIVATE.confirm	*FMA2_ACTIVATE_SAP*	*CON*	*M and S*

SAP Activate FMA1/2

FMA1/2_RSAP_ACTIVATE.request	*FMA2_ACTIVATE_RSAP*	*REQ*	*M and S*
FMA1/2_RSAP_ACTIVATE.confirm	*FMA2_ACTIVATE_RSAP*	*CON*	*M and S*

SAP Deactivate FMA1/2

FMA1/2_SAP_DEACTIVATE.request	*FMA2_DEACTIVATE_SAP*	*REQ*	*M and S*
FMA1/2_SAP_DEACTIVATE.confirm	*FMA2_DEACTIVATE_SAP*	*CON*	*M and S*

M: Master, active station
S: Slave, passive station

Individual descriptions of the above are given in the same order on the following pages.

 © 1995 PEP Modular Computers

FMA2_RESET Request

Description:

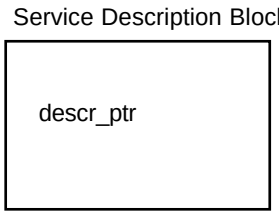
The FLC and MAC sub-layers are reinitialized. All information previously contained in FLC and MAC is lost. All data structures of the FDL-user that were contained in the FDL at the time of the reset are also lost, and the user must restore them himself.

The FDL waits for bus parameters after a *FMA2_RESET*. It also waits for bus parameters after the start of layer 2 software. Once these bus parameters are set the MAC runs and other services can be undertaken.

Important: *After the FMA2_RESET a FMA2_SET_BUSPARAMETER must be sent. No other services can be accepted until this has taken place.*

If several PROFIBUS application tasks are running, using this service will affect all PROFIBUS applications.

Data Structure:



Service Description Block:

sap	MSAP_0	Local Service Access Point
service	FMA2_RESET	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	*not significant*	
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

FMA2_RESET Confirmation

Description:

It is confirmed that the FDL has been reset.

Data Structure:

Service Description Block



Service Description Block:

sap	*remains unchanged*	
service	FMA2_RESET	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK or IV	*see below*
descr_ptr	*not significant*	
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Status Values:

Code	Meaning
OK	Positive conformation that the service has been carried out
IV	Invalid parameter in request

Note: An invalid SAP will also return 'IV' as the status.

 © 1995 PEP Modular Computers

FMA2_SET_BUSPARAMETER Request

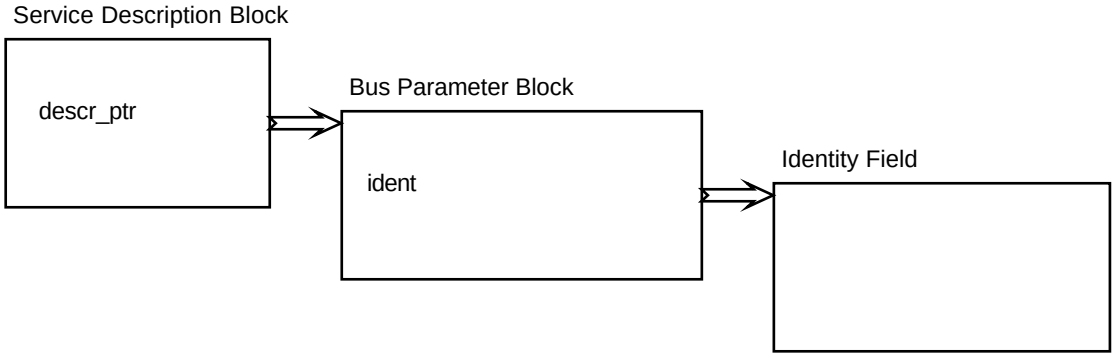
Important: *This service call must be carried out after start of layer 2 software and after every FMA2_RESET.*

Description:

The FDL-User sends the operational parameters to the FDL. The parameters *HSA*, *ident* and *in_ring_desired* are taken over after the layer 2 software starts and after every *FMA2_RESET*, and cannot be altered once in service.

If the bus parameters are set using the *FMA2_SET_BUSPARAMETER* service, further changes can be made using the *FMA2_CHANGE_BUSPARAMETER* service.

Data Structure:



All bus parameters must be transferred in the form of a bus parameter block. Since the FDL makes a copy and returns the original to the FDL-User, individual changes can be made to the original and the FDL be informed via the *FMA2_CHANGE_BUSPARAMETER* service call.

Service Description Block:

sap	MSAP_0	
service	FMA2_SET_BUSPARAMETER	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_BUSPAR_BLOCK far*)	Pointer to bus parameter block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Bus Parameter Block:

loc_add.station	0..126	Own station address
loc_add.segment	0..63 or NO_SEGMENT	Own segment address
baud_rate	K_BAUD_9_6, or K_BAUD_19_2, or K_BAUD_93_75, or K_BAUD_187_5, or K_BAUD_500	
medium_red	NO_REDUNDANCY or REDUNDANCY	
tsl	1..65535	Slot time
min_tsdr	1..65535	Minimum station delay time
max_tsdr	1..65535	Maximum station delay time
tqui	0..255	Modulator decouple time
tset	1..255	Exposure time (set up)
ttr	$1..2^{24}-1$	Target rotation time
g	1..100	GAP update factor
in_ring_desired	TRUE or FALSE	Desired participation in token ring
hsa	2..126	Highest station address (in local segment)
max_retry_limit	1..8	Max retrys in event of error
ident	(UNSIGN8 far*)	Pointer to identity field
ind_buf_len	0, 1-255	0 = token brake not active, otherwise it is active

Identity Field:

The identity field supplied by the FDL-User must contain the following "C" structure:

`UNSIGN8`	Length vendor_name
`UNSIGN8`	Length controller_type
`UNSIGN8`	Length HW_release
`UNSIGN8`	Length SW_release
`char[ident_size]`	ASCII character string

`ident_size` must be at least equal to the sum of the previous four parts, but cannot exceed 238 bytes.

Note: All times are given as bit times.

The recommended parameter values for various baud rates is shown below:

Baud Rate	**9.6**	**19.2**	**93.75**	**187.5**	**500**
tsl	*100*	*200*	*500*	*1500*	*3500*
min_tsdr	*30*	*60*	*125*	*250*	*500*
max_tsdr	*50*	*100*	*250*	*500*	*1000*
tqui	*0*	*0*	*0*	*0*	*0*
tset	*5*	*10*	*15*	*25*	*50*
ttr	*10000*	*15000*	*30000*	*50000*	*100000*
g	*1*	*1*	*1*	*1*	*1*

The target rotation time (*ttr*) depends upon the actual load and number of connected stations.Too small a value should not be chosen for the sake of efficiency.

Layer 2 demands on the processor can be reduced by using a large slot time and a small GAP factor. This will ensure that not much time is wasted waiting for responses from busy or non-available partners.

The participant is passive when the *in_ring_desired* is set to FALSE.

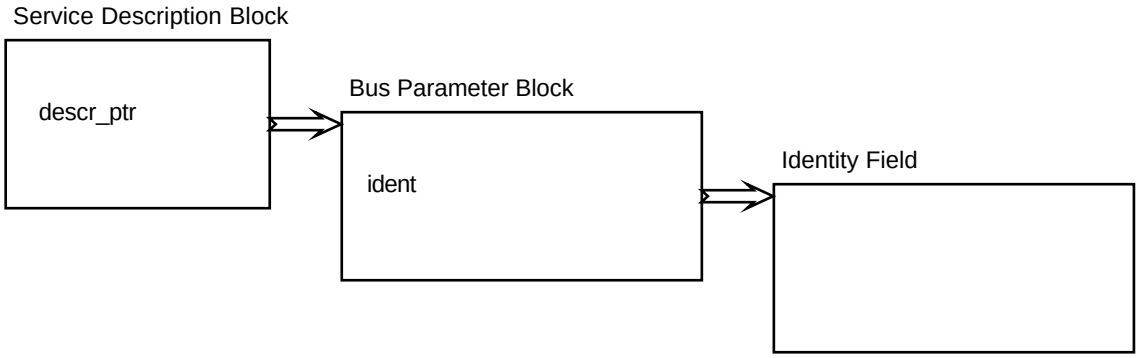
 © 1995 PEP Modular Computers

FMA2_SET_BUSPARAMETER Confirmation

Description:

The acceptance of the bus parameters are confirmed or, if errornous (invalid parameters, etc.), denied.

Data Structure:



The bus parameter block is returned to the FDL-User. The FDL keeps a copy of the data. It is recommended that the FDL-User retains the bus parameter block for any minor changes needed later.

Service Description Block:

sap	*remains unchanged*	
service	FMA2_SET_BUSPARAMETER	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, IV or LR	*see below*
descr_ptr	*remains unchanged*	Pointer to bus parameter block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Status Values:

Code	Meaning
OK	Positive conformation that the service has been carried out
IV	Invalid parameter in request
LR	Bus parameters have already been set using a previous *FMA2_SET_BUSPARAMETER* service

 © 1995 PEP Modular Computers

FMA2_CHANGE_BUSPARAMETER Request

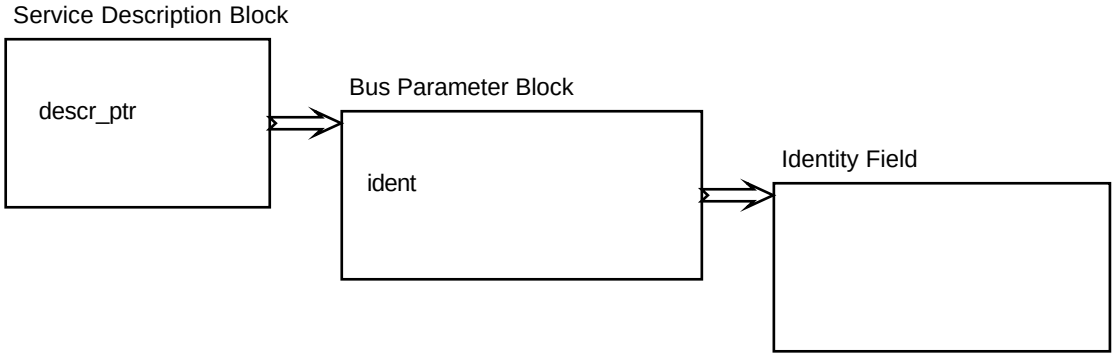
Important: *This service call can be carried out after a FMA2_SET_BUSPARAMETER service.*

Description:

The FDL-User sends the operational parameters to the FDL. The parameters *HSA*, *ident* and *in_ring_desired* are taken over after the layer 2 software starts and after every *FMA2_RESET*, and cannot be altered once in service.

After every call for *FMA2_CHANGE_BUSPARAMETER*, the MAC leaves the logical token ring and takes up an active or passive idle mode according to the *in_ring_desired* parameter.

Data Structure:



All bus parameters must be transferred in the form of a bus parameter block. Since the FDL makes a copy and returns the original to the FDL-User, individual changes can be made to the original and the FDL be informed via the *FMA2_CHANGE_BUSPARAMETER* service call.

Service Description Block:

sap	MSAP_0	
service	FMA2_CHANGE_BUSPARAMETER	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_BUSPAR_BLOCK far*)	Pointer to bus parameter block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Bus Parameter Block:

loc_add.station	0..126	Own station address
loc_add.segment	0..63 or NO_SEGMENT	Own segment address
baud_rate	K_BAUD_9_6, or K_BAUD_19_2, or K_BAUD_93_75, or K_BAUD_187_5, or K_BAUD_500	
medium_red	NO_REDUNDANCY or REDUNDANCY	
tsl	1..65535	Slot time
min_tsdr	1..65535	Minimum station delay time
max_tsdr	1..65535	Maximum station delay time
tqui	0..255	Modulator decouple time
tset	1..255	Exposure time (set up)
ttr	$1..2^{24}-1$	Target rotation time
g	1..100	GAP update factor
in_ring_desired	TRUE or FALSE	Desired participation in token ring
hsa	2..126	Highest station address (in local segment)
max_retry_limit	1..8	Max retrys in event of error
ident	(UNSIGN8 far*)	Pointer to identity field
ind_buf_len	0, 1-255	0 = token brake not active, otherwise it is active

Identity Field:

The identity field supplied by the FDL-User must contain the following "C" structure:

`UNSIGN8`	Length vendor_name
`UNSIGN8`	Length controller_type
`UNSIGN8`	Length HW_release
`UNSIGN8`	Length SW_release
`char[ident_size]`	ASCII character string

`ident_size` must be at least equal to the sum of the previous four parts, but cannot exceed 238 bytes.

Note: All times are given as bit times.

The recommended parameter values for various baud rates is shown below:

Baud Rate	**9.6**	**19.2**	**93.75**	**187.5**	**500**
tsl	*100*	*200*	*500*	*1500*	*3500*
min_tsdr	*30*	*60*	*125*	*250*	*500*
max_tsdr	*50*	*100*	*250*	*500*	*1000*
tqui	*0*	*0*	*0*	*0*	*0*
tset	*5*	*10*	*15*	*25*	*50*
ttr	*10000*	*15000*	*30000*	*50000*	*100000*
g	*1*	*1*	*1*	*1*	*1*

The target rotation time (*ttr*) depends upon the actual load and number of connected stations.Too small a value should not be chosen for the sake of efficiency.

Layer 2 demands on the processor can be reduced by using a large slot time and a small GAP factor. This will ensure that not much time is wasted waiting for responses from busy or non-available partners.

The participant is passive when the *in_ring_desired* is set to FALSE.

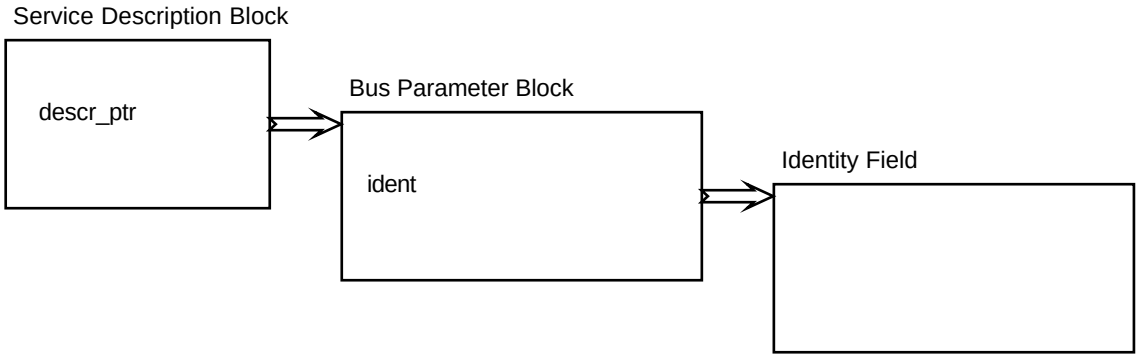
 © 1995 PEP Modular Computers

FMA2_CHANGE_BUSPARAMETER Confirmation

Description:

The acceptance of the bus parameters are confirmed or, if errornous (invalid parameters, etc.), denied.

Data Structure:



The bus parameter block is returned to the FDL-User. The FDL keeps a copy of the data. It is recommended that the FDL-User retains the bus parameter block for any minor changes needed later.

Service Description Block:

sap	*remains unchanged*	
service	FMA2_CHANGE_BUSPARAMETER	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, IV or LR	*see below*
descr_ptr	*remains unchanged*	Pointer to bus parameter block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Status Values:

Code	Meaning
OK	Positive conformation that the service has been carried out
IV	Invalid parameter in request or invalid service
LR	Bus parameters are not set by a previous FMA2_SET_BUSPARAMETER service

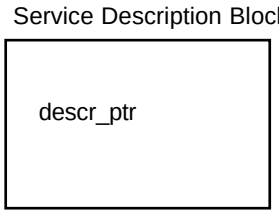
 © 1995 PEP Modular Computers

FMA2_SET_STATISTIC_CTR Request

Description:

The statistic counter in the FDL is reset to zero and restarted.

Data Structure:



Service Description Block:

sap	MSAP_0	
service	FMA2_SET_STATISTIC_CTR	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	*not significant*	
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

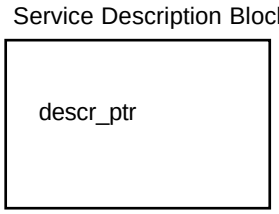
 © 1995 PEP Modular Computers

FMA2_SET_STATISTIC_CTR Confirmation

Description:

The reset of the statistical counter is confirmed.

Data Structure:



Service Description Block:

sap	*remains unchanged*	
service	FMA2_SET_STATISTIC_CTR	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK or IV	
descr_ptr	*not significant*	
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Status Values:

Code	Meaning
OK	Positive conformation that the service has been carried out
IV	Invalid parameter in request

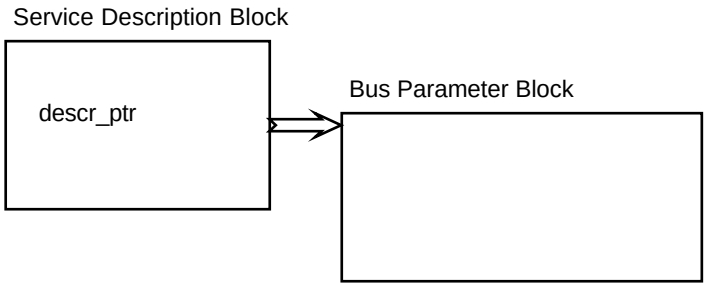
 © 1995 PEP Modular Computers

FMA2_READ_BUSPARAMETER Request

Description:

This primitive is used by the FDL to read the actual bus parameters with the exception of the identity field. The *FMA2_IDENT* service is used to read the identity field.

Data Structure:



The FDL-User supplies a structure of the type *T_FDL_BUSPAR_BLOCK* to the FDL. No additional reference must be made for the identity field.

Service Description Block:

sap	MSAP_0	
service	FMA2_READ_BUSPARAMETER	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_BUSPAR_BLOCK far *)	Pointer to bus parameter block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Bus Parameter Block:

The bus parameter block must have the same structure as in the *FMA2_SET_BUSPARAMETER* service call. The pointer to the identity field is however, without significance.

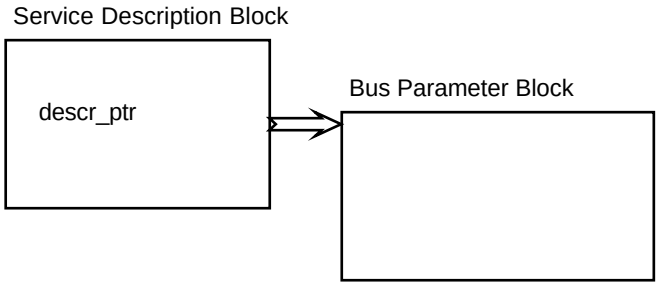
 © 1995 PEP Modular Computers

FMA2_READ_BUSPARAMETER Confirmation

Description:

The read bus parameters are given over to the FDL-User.

Data Structure:



Service Description Block:

sap	*remains unchanged*	
service	FMA2_SET_BUSPARAMETER	
primitive	REQ	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK or IV	*see below*
descr_ptr	*remains unchanged*	
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Bus Parameter Block:

loc_add.station	0..126	Own station address
loc_add.segment	0..63 or NO_SEGMENT	Own segment address
baud_rate	K_BAUD_9_6, or K_BAUD_19_2, or K_BAUD_93_75, or K_BAUD_187_5, or K_BAUD_500	
medium_red	NO_REDUNDANCY or REDUNDANCY	
tsl	1..65535	Slot time
min_tsdr	1..65535	Minimum station delay time
max_tsdr	1..65535	Maximum station delay time
tqui	0..255	Modulator decouple time
tset	1..255	Exposure time (set up)
ttr	$1..2^{24}-1$	Target rotation time
g	1..100	GAP update factor
in_ring_desired	TRUE or FALSE	Desired participation in token ring
hsa	2..126	Highest station address (in local segment)
max_retry_limit	1..8	Maximum retrys in event of error
ident	NULL	
ind_buf_len	0, 1-255	Token brake

 © 1995 PEP Modular Computers

Status Values:

Code	Meaning
OK	Positive conformation that the service has been carried out
IV	Invalid parameter in request

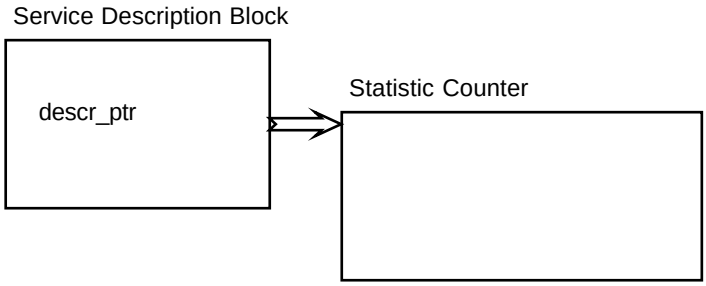
 © 1995 PEP Modular Computers

FMA2_READ_STATISTIC_CTR Request

Description:

With this service the FDL is instructed to read the statistical counter.

Data Structure:



The FDL-User must use a data structure of the type *T_FDL_STATISTIC_CTR* into which the read values are entered.

Service Description Block:

sap	MSAP_0	
service	FMA2_READ_STATISTIC_CTR	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_STATISTIC_CTR far *)	Pointer to statistic counter
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Statistic Counter:

frame_send_count	*not significant*
retry_count	*not significant*
sd_count	*not significant*
sd_error_count	*not significant*

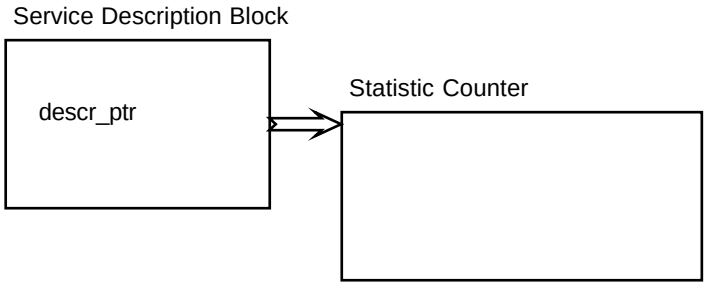
 © 1995 PEP Modular Computers

FMA2_READ_STATISTIC_CTR Confirmation

Description:

The read statistical values are transferred to the FDL-User.

Data Structure:



Service Description Block:

sap	*remains unchanged*	
service	FMA2_READ_STATISTIC_CTR	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK or IV	Status
descr_ptr	*remains unchanged*	Pointer to statistic counter
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Statistic Counter:

frame_send_count	$1..2^{32}-1$	Number of sent telegrams
retry_count	$1..2^{16}-1$	Number of repeated telegrams
sd_count	$1..2^{32}-1$	Number of valid start delimiters
sd_error_count	$1..2^{16}-1$	Number of errornous start delimiters

Status Values:

Code	Meaning
OK	Positive conformation that the service has been carried out
IV	Invalid parameter in request

 © 1995 PEP Modular Computers

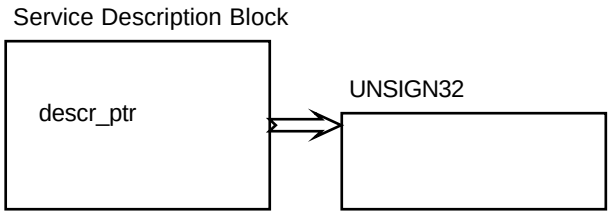
FMA2_READ_TRR Request

Description:

With this service the FDL is given the task of reading the "Real Target Rotation Time".

Note: This service is only supported on active (Master) stations.

Data Structure:



The FDL-User must supply a pointer to the 32-bit variable into which the FDL can place the read value.

Service Description Block:

sap	MSAP_0	
service	FMA2_READ_TRR	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(UNSIGN32 far *)	Pointer to 32-bit variable
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

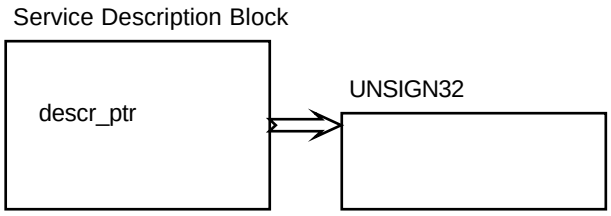
FMA2_READ_TRR Confirmation

Description:

The read "Real Target Rotation Time" is given to the FDL-User.

Note: This service is only supported on active (Master) stations.

Data Structure:



Service Description Block:

sap	*remains unchanged*	
service	FMA2_READ_TRR	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, IV or NO	*see below*
descr_ptr	(UNSIGN32 far *)	Pointer to TRR variable
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Status Values:

Code	Meaning
OK	Positive conformation that the service has been carried out
NO	No reply data will be transfered
IV	Invalid parameter in request

 © 1995 PEP Modular Computers

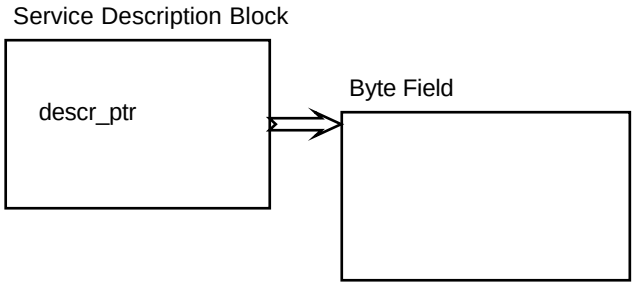
FMA2_READ_LAS Request

Description:

The FDL is instructed to read the "List of Active Stations" (LAS) This service is only supported by the active participants.

Note: This service is only supported on active (Master) stations.

Data Structure:



The FDL-User points to a byte field with the length *hsa+1* into which the FDL can enter the LAS.

Service Description Block:

sap	MSAP_0	
service	FMA2_READ_LAS	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(UNSIGN8 far *)	Pointer to byte field (length *hsa+1*)
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

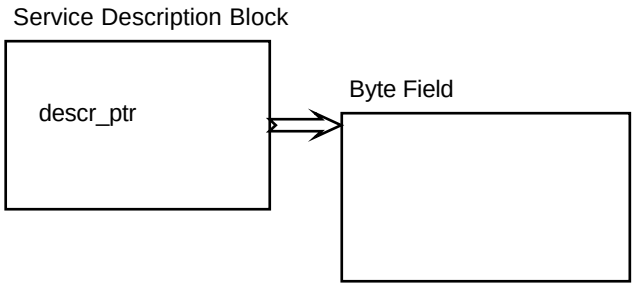
FMA2_READ_LAS Confirmation

Description:

The read value of the "List of Active Stations" (LAS) is given to the FDL-User.

Note: This service is only supported on active (Master) stations.

Data Structure:



Service Description Block:

sap	*remains unchanged*	
service	FMA2_READ_LAS	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, IV or NO	Status
descr_ptr	(UNSIGN8 far *)	Pointer to read LAS
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

List of Active Stations (LAS):

The LAS is entered into the byte field, given during the request.

Byte i (0 < = i < = hsa) gives the status of participant i.

00 = Participant is not active in the logical token ring.

01 = Participant is active in the logical token ring.

Status Values:

Code	Meaning
OK	Positive conformation that the service has been carried out
NO	No reply data will be transfered
IV	Invalid parameter in request

 © 1995 PEP Modular Computers

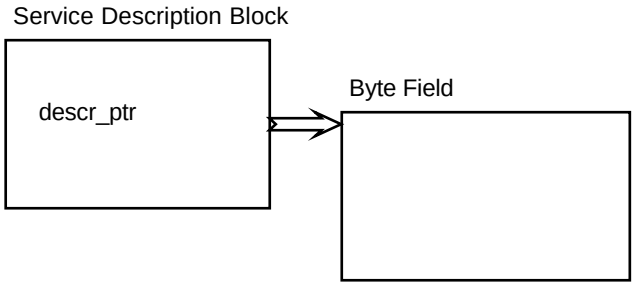
FMA2_READ_GAPLIST Request

Description:

The FDL is instructed to read the "GAP-List". This service lists stations that lie between the next active station and your own address.

Note: This service is only supported on active (Master) stations.

Data Structure:



The FDL-User points to a byte field with the length *hsa+1* into which the FDL can enter the "GAP-List".

Service Description Block:

sap	MSAP_0	
service	FMA2_READ_GAPLIST	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(UNSIGN8 far *)	Pointer to byte field (length *hsa+1*)
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

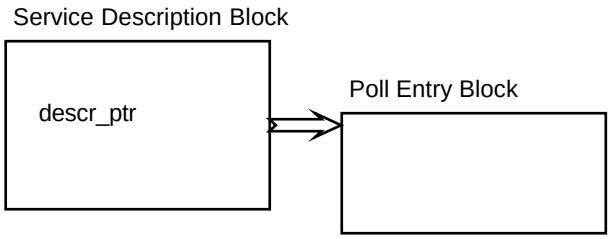
FMA2_READ_GAPLIST Confirmation

Description:

The "GAP-List" is given over to the FDL-User.

Note: This service is only supported on active (Master) stations.

Data Structure:



Service Description Block:

sap	MSAP_0	
service	FMA2_READ_GAPLIST	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, IV or NO	Status
descr_ptr	(UNSIGN8 far *)	Pointer to byte field (length *hsa + 1*)
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

GAP-List:

The GAP-List is entered into the byte field, given during the request. It only provides information about the gaps between the addresses of the other active participants and yourself.

Byte i (0 < = i < = hsa) gives the status of participant i.

00 = Passive participant.

01 = Active participant, not ready for the logical token ring.

02 = Active participant, ready for the logical token ring.

03 = Active participant, currently on the logical token ring.

17 = Participant unknown, no answer.

255 = Participant is not in own GAP area.

Status Values:

Code	Meaning
OK	Positive conformation that the service has been carried out
NO	No reply data will be transfered
IV	Invalid parameter in request

 © 1995 PEP Modular Computers

FMA2_EVENT Indication

Description:

The FDL informs the FDL-User that an event or an error has occurred.

Data Structure:

Service Description Block



Service Description Block:

sap	MSAP_1	
service	FMA2_EVENT	
primitive	IND	
user_id	0..65535	Identification possibility for FDL-User
status	*see below*	Event or error
descr_ptr	*not significant*	
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Status Values:

1	FMA2_FAULT_ADDRESS	Multiple FDL addresses exist for this participant
2	FMA2_FAULT_TRANSCEIVER	Error in transmitter or receiver
3	FMA2_FAULT_TTO	Bus timeout, T_{TO} expired
4	FMA2_FAULT_SYN	No receiving synchronization, T_{SYNI} expired
5	FMA2_FAULT_OUT_OF_RING	An active participant has left the logical ring
6	FMA2_GAP_EVENT	A new participant has been inserted into or removed from the GAP area of the token ring

 © 1995 PEP Modular Computers

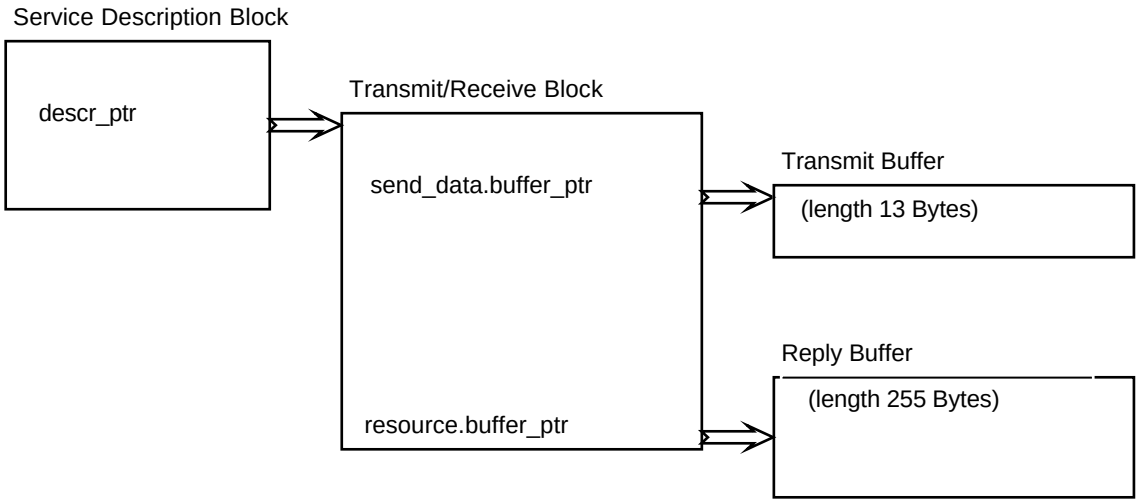
FMA2_IDENT Request

Description:

The FDL is given the task of reading its own identification, or that of another participant.

Note: Requesting the identification from remote stations is only supported by active stations.

Data Structure:



The FDL-User must provide a pointer to the transmit/receive block. This in turn points to the buffers containing the call and answer telegrams. The buffer for the calling telegram must be 13 bytes long and that of the answer telegram 255 bytes (in order to accept the longest identification length).

Service Description Block:

sap	MSAP_0	
service	FMA2_IDENT	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_SR_BLOCK far*)	Pointer to transmit/receive block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Transmit/Receive Block:

loc_add.station	*not significant*	
loc_add.segment	*not significant*	
remote_sap	*not significant*	
rem_add.station	0..126	Address of station to be identified
rem_add.segment	NO_SEGMENT	FMA2 service only in own segment
serv_class	*not significant*	
update_status	*not significant*	
send_data.buffer_ptr	(UNSIGN8 far*)	Pointer to transmit buffer
send_data.length	*not significant*	Length of user data
receive_data.buffer_ptr	*not significant*	
receive_data.length	*not significant*	
resource.buffer_ptr	(UNSIGN8 far*)	Pointer to reply buffer
resource.length	255	Length of reply buffer

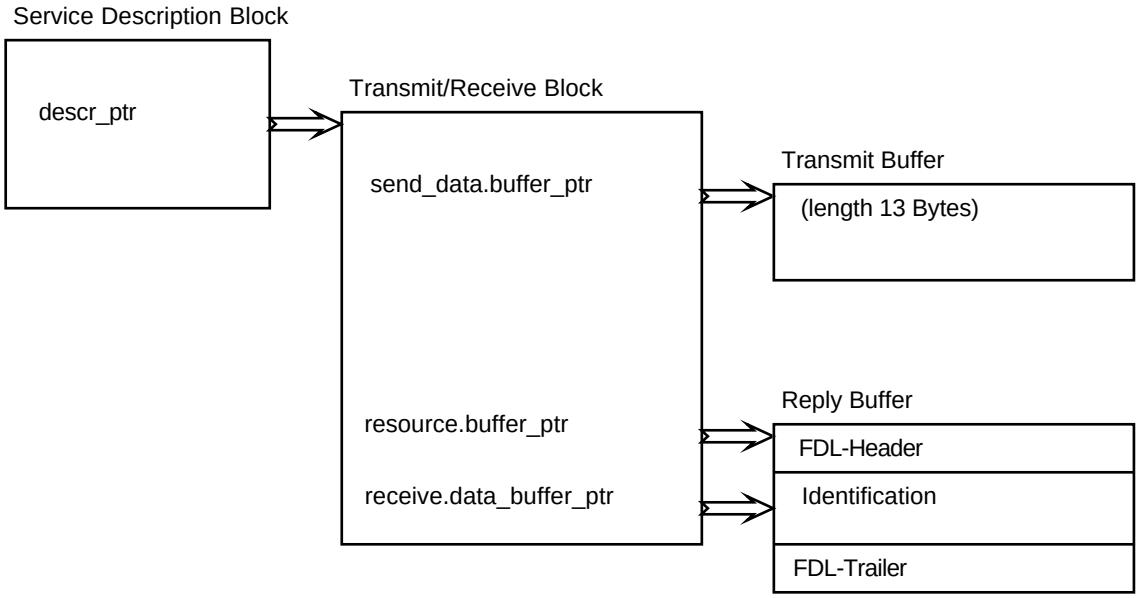
 © 1995 PEP Modular Computers

FMA2_IDENT Confirmation

Description:

The FDL receives the requested identification or returns a negative status.

Data Structure:



The pointer *receive.buffer_ptr* shows the start of the identity field in the reply buffer.

Service Description Block:

sap	*remains unchanged*	
service	FMA2_IDENT	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, LR, NA, NLT, NR or IV	*see below*
descr_ptr	*remains unchanged*	Pointer to transmit/receive block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Transmit/Receive Block:

loc_add.station	*not significant*	
loc_add.segment	*not significant*	
remote_sap	*not significant*	
rem_add.station	*remains unchanged*	
rem_add.segment	*remains unchanged*	
serv_class	*not significant*	
update_status	*not significant*	
send_data.buffer_ptr	*remains unchanged*	Pointer to transmit buffer
send_data.length	*remains unchanged*	
receive_data.buffer_ptr	(USIGN8 far *)	Pointer to identity field buffer
receive_data.length	4..242	Length of identity data
resource.buffer_ptr	*remains unchanged*	
resource.length	*remains unchanged*	

Identity Field:

The identification is placed into a field (identity field) having the following structure:

UNSIGN8	Length vendor_name
UNSIGN8	Length controller_type
UNSIGN8	Length HW_release
UNSIGN8	Length SW_release
char[238]	ASCII character string

Status Values:

Code	Meaning
OK	Identification could take place
LR	Local resource limitation
NTL	Own station not in logical token ring
NA	The called participant did not answer
NR	The identity data is not available at the called station
IV	Invalid parameter in request

 © 1995 PEP Modular Computers

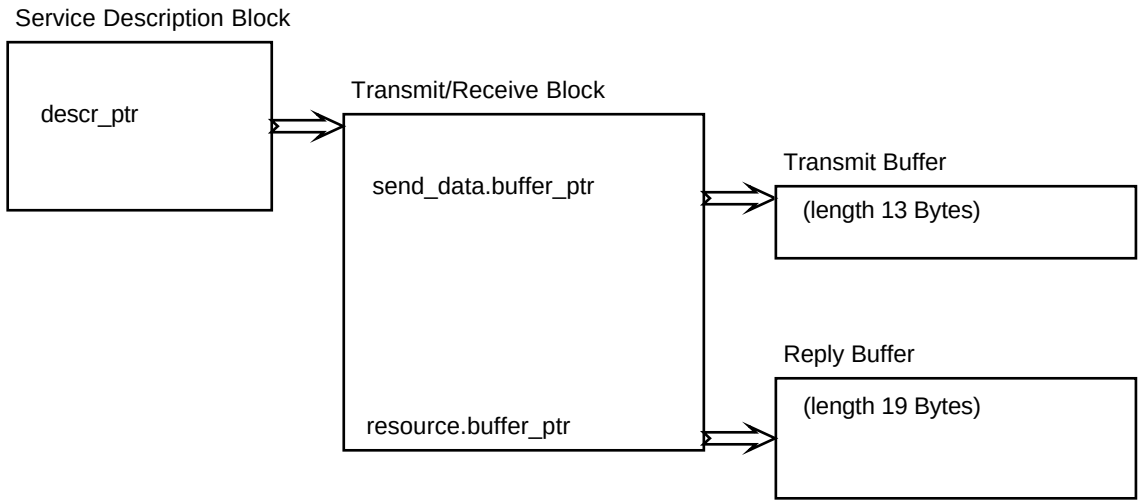
FMA2_LSAP_STATUS Request

Description:

The FDL is given the task of supplying its SAP configuration, or determining that of another participant. Only active participants support a request for SAP configuration.

Note: Requesting the SAP configuration from a remote station is only supported by active stations.

Data Structure:



The FDL-User must provide a pointer to the transmit/receive block. This in turn points to the buffers containing the call and answer telegrams. The buffer for the transmit telegram must be 13 bytes long and that of the answer telegram 19 bytes (to accept the 6 byte status information).

Service Description Block:

sap	MSAP_0	
service	FMA2_LSAP_STATUS	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_SR_BLOCK far*)	Pointer to transmit/receive block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Transmit/Receive Block:

loc_add.station	*not significant*	
loc_add.segment	*not significant*	
remote_sap	0..63 or DAULT_SAP	Desired Service Access Point
rem_add.station	0..126	Address of station desired
rem_add.segment	NO_SEGMENT	FMA2 service only in own Segment
serv_class	*not significant*	
update_status	*not significant*	
send_data.buffer_ptr	(UNSIGN8 far*)	Pointer to transmit buffer
send_data.length	*not significant*	Length of user data
receive_data.buffer_ptr	*not significant*	
receive_data.length	*not significant*	
resource.buffer_ptr	(UNSIGN8 far*)	Pointer to reply buffer
resource.length	19	Length of reply buffer

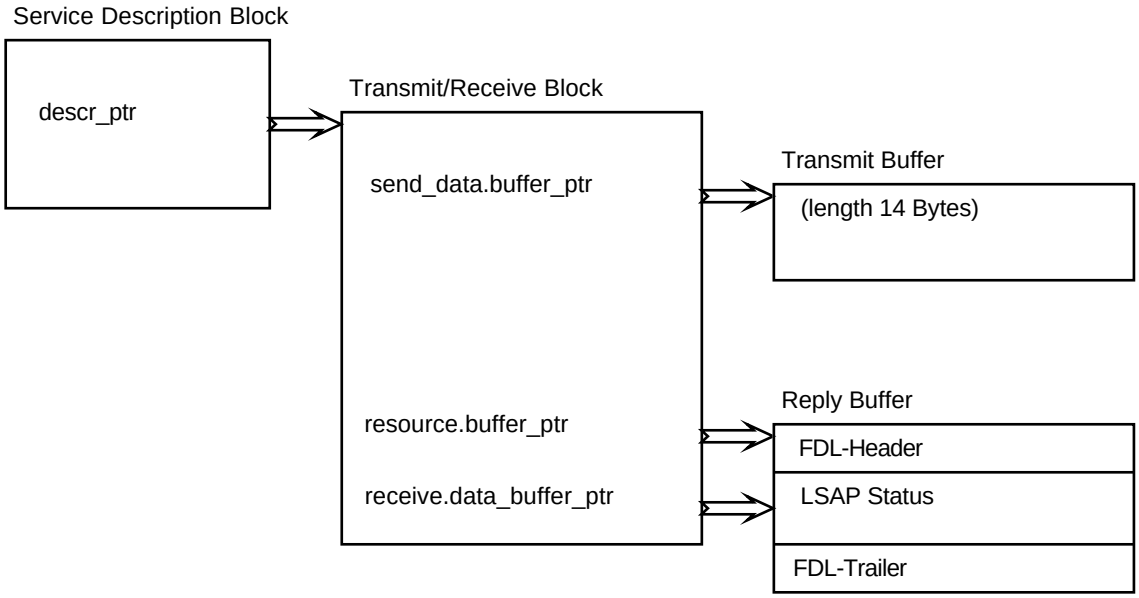
 © 1995 PEP Modular Computers

FMA2_LSAP_STATUS Confirmation

Description:

The FDL receives the requested configuration data or returns a negative status.

Data Structure:



If the status is OK, the pointer *receive.buffer_ptr* shows the start of the status field in the reply buffer.

Service Description Block:

sap	*remains unchanged*	
service	FMA2_LSAP_STATUS	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, RS, NA, NLT, NR or IV	*see below*
descr_ptr	*remains unchanged*	Pointer to transmit/receive block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Transmit/Receive Block:

loc_add.station	*not significant*	
loc_add.segment	*not significant*	
remote_sap	*remains unchanged*	
rem_add.station	*remains unchanged*	
rem_add.segment	*remains unchanged*	
serv_class	*not significant*	
update_status	*not significant*	
send_data.buffer_ptr	*remains unchanged*	Pointer to transmit buffer
send_data.length	*remains unchanged*	
receive_data.buffer_ptr	(USIGN8 far *)	Pointer to LSAP status field
receive_data.length	6	Length of LSAP status field
resource.buffer_ptr	*remains unchanged*	Pointer to reply buffer
resource.length	*remains unchanged*	Length of reply buffer

LSAP Status Field:

The identification is placed into a field (LSAP status field) having the following structure:

UNSIGN8	access.station	0..126 or global 127
UNSIGN8	access.segment	0..63 or NO_SEGMENT
UNSIGN8	service 1	*see below*
UNSIGN8	service 2	*see below*
UNSIGN8	service 3	*see below*
UNSIGN8	service 4	*see below*

For the parameters service 1 to service 4 the following coding is used:

B7	B6	B5	B4	Service	B3	B2	B1	B0	Role
0	*0*	*0*	*0*	*SDA*	*0*	*0*	*0*	*0*	*Initiator*
0	*0*	*0*	*1*	*SDN*	*0*	*0*	*0*	*1*	*Responder*
0	*0*	*1*	*1*	*SRD*	*0*	*0*	*1*	*0*	*Both*
0	*1*	*0*	*1*	*CSRD*	*0*	*0*	*1*	*1*	*Not activated*

Status Values:

Code	Meaning
OK	Status reading could take place
RS	Addressed participants SAP not active
NTL	Own station not in logical ring
NA	The called participant did not answer
NR	The status data is not available at the called station
IV	Invalid parameters in request

 © 1995 PEP Modular Computers

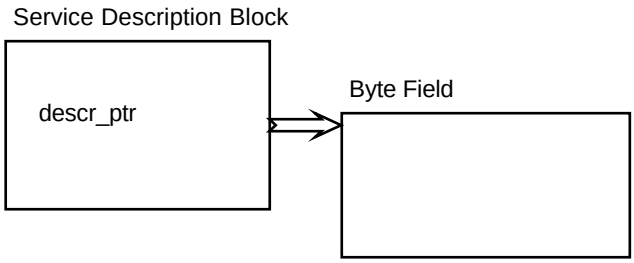
FMA2_LIVELIST Request

Description:

The FDL is task with reading a "Live-List", i.e. a list of participants currently active on the bus.

Note: This service is only supported by active (Master) stations.

Data Structure:



The FDL-User points to a byte field with the length *HSA+1 = 127* into which the FDL can enter the Live-List.

Service Description Block:

sap	MSAP_0	
service	FMA2_LIVELIST	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(UNSIGN8 far*)	Pointer to byte field *HSA+1*
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

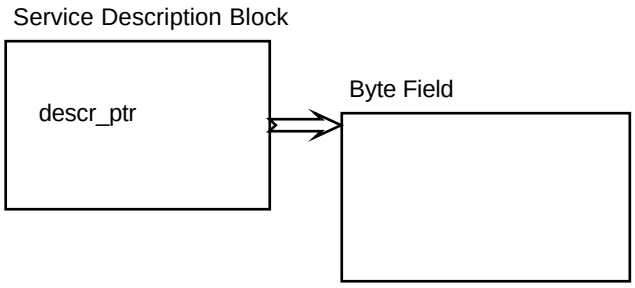
FMA2_LIVELIST Confirmation

Description:

The "Live-List" or a negative status is given to the FDL-User.

Note: This service is only supported by active (Master) stations.

Data Structure:



Service Description Block:

sap	*remains unchanged*	
service	FMA2_READ_LIVELIST	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, LR, NLT or IV	*see below*
descr_ptr	*remains unchanged*	Pointer to read Live-List
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Live-List:

The Live-List is entered into the byte field, given during the request.

Byte i (0 < = i < = 126) gives the status of participant i.

00 = Passive participant.

01 = Active participant, not ready for the logical token ring.

02 = Active participant, ready for the logical token ring.

03 = Active participant, currently on the logical token ring.

17 = Participant unknown, no answer.

Status Values:

Code	Meaning
OK	Positive conformation that the service has been carried out
LR	None or insufficient operational resources are available locally
NLT	Local partner not in logical ring or has left bus
IV	Invalid parameter in request

 © 1995 PEP Modular Computers

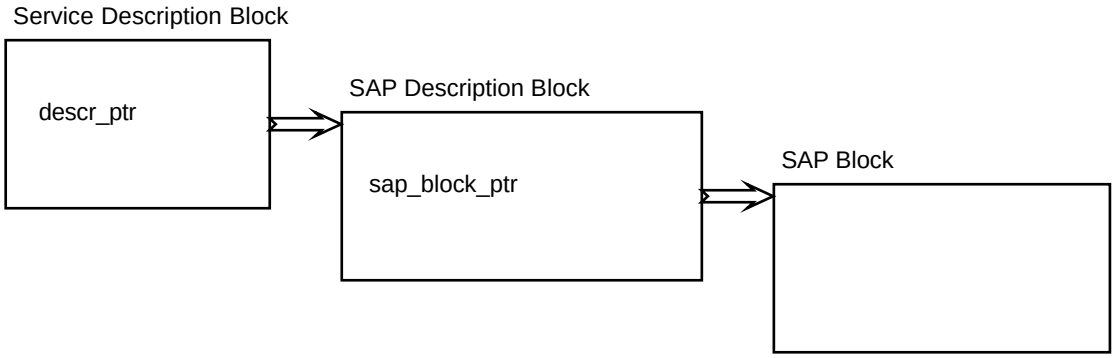
FMA2_ACTIVATE_SAP Request

Description:

The local Service Access Point (SAP) is activated and configured.

If an SAP for response functions of the SRD or CSRD service is required, the SAP is configured via the *FMA2_ACTIVATE_RSAP* service call.

Data Structure:



The FDL-User supplies a pointer to the SAP description block of type *T_FDL_SAP_DESCR*, this in turn points to a SAP block of type *T_FDL_SAP_BLOCK*. The SAP description block and the SAP block remain in the FDL until the service point is deactivated.

Service Description Block:

sap	MSAP_2	
service	FMA2_ACTIVATE_SAP	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_SAP_DESCR far *)	Pointer to SAP description block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

SAP Description Block (T_FDL_SAP_DESCR):

sap_nr	0..63 or DEFAULT_SAP	SAP to be activated
rem_add.station	0..126 or global address 127	Access protection for responder function
rem_add.segment	0..63 or NO_SEGMENT	Access protection for responder function
sda	INITIATOR or RESPONDER or BOTH_ROLES or SERVICE_NOT_ACTIVATED	
sdn	INITIATOR or RESPONDER or BOTH_ROLES or SERVICE_NOT_ACTIVATED	
srd	INITIATOR or SERVICE_NOT_ACTIVATED	
csrd	INITIATOR or SERVICE_NOT_ACTIVATED	
services	*reserved for FDL*	
sap_block_ptr	(T_FDL_SAP_BLOCK far*)	
resrc_ptr	*not significant*	
resrc_ctr	*not significant*	
sema	*reserved for FDL*	

 © 1995 PEP Modular Computers

SAP Block (T_FDL_SAP_BLOCK):

max_len_sda_req_low	0..242	Max. length of user data by SDA.req low
max_len_sda_req_high	0..242	Max. length of user data by SDA.req high
max_len_sda_ind_low	0..242	Max. length of user data by SDA.ind low
max_len_sda_ind_high	0..242	Max. length of user data by SDA.ind high
max_len_sdn_req_low	0..242	Max. length of user data by SDN.req low
max_len_sdn_req_high	0..242	Max. length of user data by SDN.req high
max_len_sdn_ind_low	0..242	Max. length of user data by SDN.ind low
max_len_sdn_ind_high	0..242	Max. length of user data by SDN.ind high
max_len_srd_req_low	0..242	Max. length of user data by SRD.req low
max_len_srd_req_high	0..242	Max. length of user data by SRD.req high
max_len_srd_con_low	0..242	Max. length of user data by SRD.con low
max_len_srd_con_high	0..242	Max. length of user data by SRD.con high

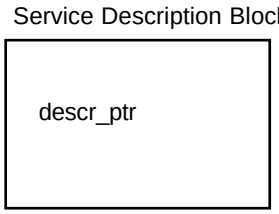
 © 1995 PEP Modular Computers

FMA2_ACTIVATE_SAP Confirmation

Description:

The FDL confirms activation of the Service Access Point, or an error-status is returned.

Data Structure:



In the event of an error, the data structure received during the request is returned. A positive confirmation (an "OK" status) results in only the Service Description Block being returned.

Service Description Block:

sap	*remains unchanged*	
service	FMA2_ACTIVATE_SAP	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, NO or IV	*see below*
descr_ptr	NULL	By error remains unchanged
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Status Values:

Code	Meaning
OK	The SAP could be accessed in the desired way
NO	The SAP could not be activated, or is already active
IV	Invalid parameter in request

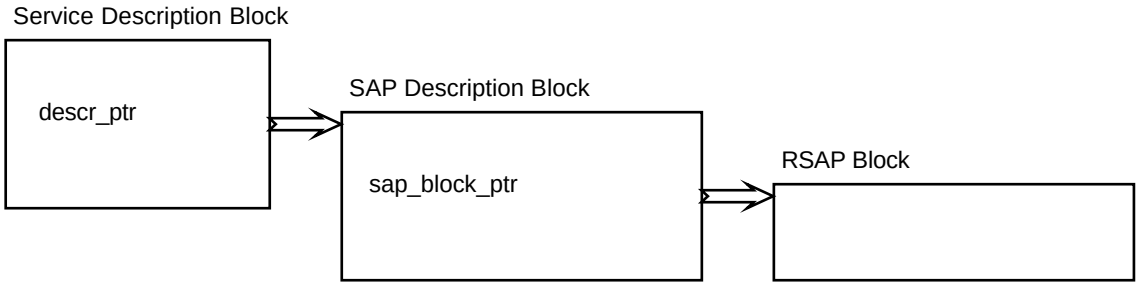
 © 1995 PEP Modular Computers

FMA2_ACTIVATE_RSAP Request

Description:

A local Service Access Point is activated and configured for a responder function with the SRD and CSRD services.

Data Structure:



The FDL-User supplies a pointer to the SAP description block of the *T_FDL_SAP_DESCR* type, this in turn points to a RSAP block of the *T_FDL_RSAP_BLOCK* type. The SAP description block and the RSAP block remain in the FDL until the service point is deactivated.

Service Description Block:

sap	MSAP_2	
service	FMA2_ACTIVATE_RSAP	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_SAP_DESCR far*)	Pointer to SAP description block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

SAP Description Block (T_FDL_SAP_DESCR):

sap_nr	0..63 or DEFAULT_SAP	SAP to be activated
rem_add.station	0..126 or global address 127	Access protection during responder function
rem_add.segment	0..63 or NO_SEGMENT	Access protection during responder function
sda	SERVICE_NOT_ACTIVATED	
sdn	SERVICE_NOT_ACTIVATED	
srd	RESPONDER	
csrd	SERVICE_NOT_ACTIVATED	
services	*reserved for FDL*	
sap_block_ptr	(T_FDL_RSAP_BLOCK far*)	
resrc_ptr	*not significant*	
resrc_ctr	*not significant*	
sema	*reserved for FDL*	

 © 1995 PEP Modular Computers

RSAP Block (T_FDL_RSAP_BLOCK):

indication_mode	ALL or DATA	
max_len_upd_req_low	0..242	Max. length of user data by REPLY_UPDATE.req low
max_len_upd_req_high	0..242	Max. length of user data by REPLY_UPDATE.req high
max_len_sdr_ind_low	0..242	Max. length of user data by SRD.ind low
max_len_sdr_ind_high	0..242	Max. length of user data by SRD.ind high
upd_buf_low	*not significant*	
upd_buf_high	*not significant*	
telegram_low	*not significant*	
telegram_high	*not significant*	
transmit_low	*reserved for FDL*	
transmit_high	*reserved for FDL*	
marker_low	*reserved for FDL*	
marker_high	*reserved for FDL*	
fcs_low	*reserved for FDL*	
fcs_high	*reserved for FDL*	

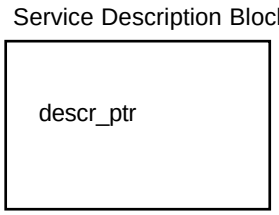
 © 1995 PEP Modular Computers

FMA2_ACTIVATE_RSAP Confirmation

Description:

The local station's FDL confirms activation of the Service Access Point for a responder function under SRD or CSRD services, or returns an error status.

Data Structure:



In the event of an error, the data structure received during the request is returned. A positive confirmation (an "OK" status) results in only the Service Description Block being returned.

Service Description Block:

sap	*remains unchanged*	
service	FMA2_ACTIVATE_RSAP	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, NO or IV	*see below*
descr_ptr	NULL	In event of error remains unchanged
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Status Values:

Code	Meaning
OK	The Service Access Point could be activated in the desired way
NO	The Service Access Point could not be activated in the desired way or is already activated
IV	Invalid parameter in request

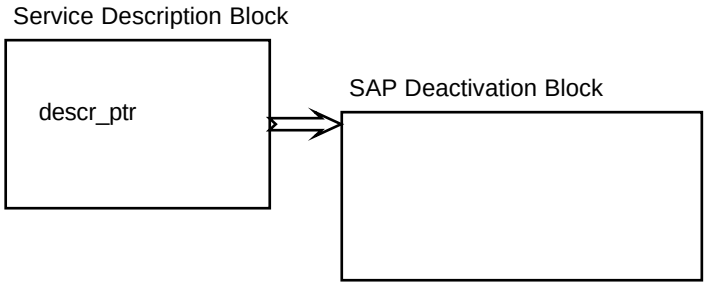
 © 1995 PEP Modular Computers

FMA2_DEACTIVATE_SAP Request

Description:

The FDL has the task of deactivating a local Service Access Point.

Data Structure:



The FDL-User provides a pointer to the SAP deactivation block, in which the Service Access Point to be deactivated is defined.

Service Description Block:

sap	MSAP_2	
service	FMA2_DEACTIVATE_SAP	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_DEACT_SAP far *)	Pointer to SAP deactivation block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

SAP Deactivation Block (T_FDL_DEACT_SAP)

ssap	0..63 or DEFAULT_SAP	Service Access Point to be deactivated
sap_descr_ptr	*not significant*	

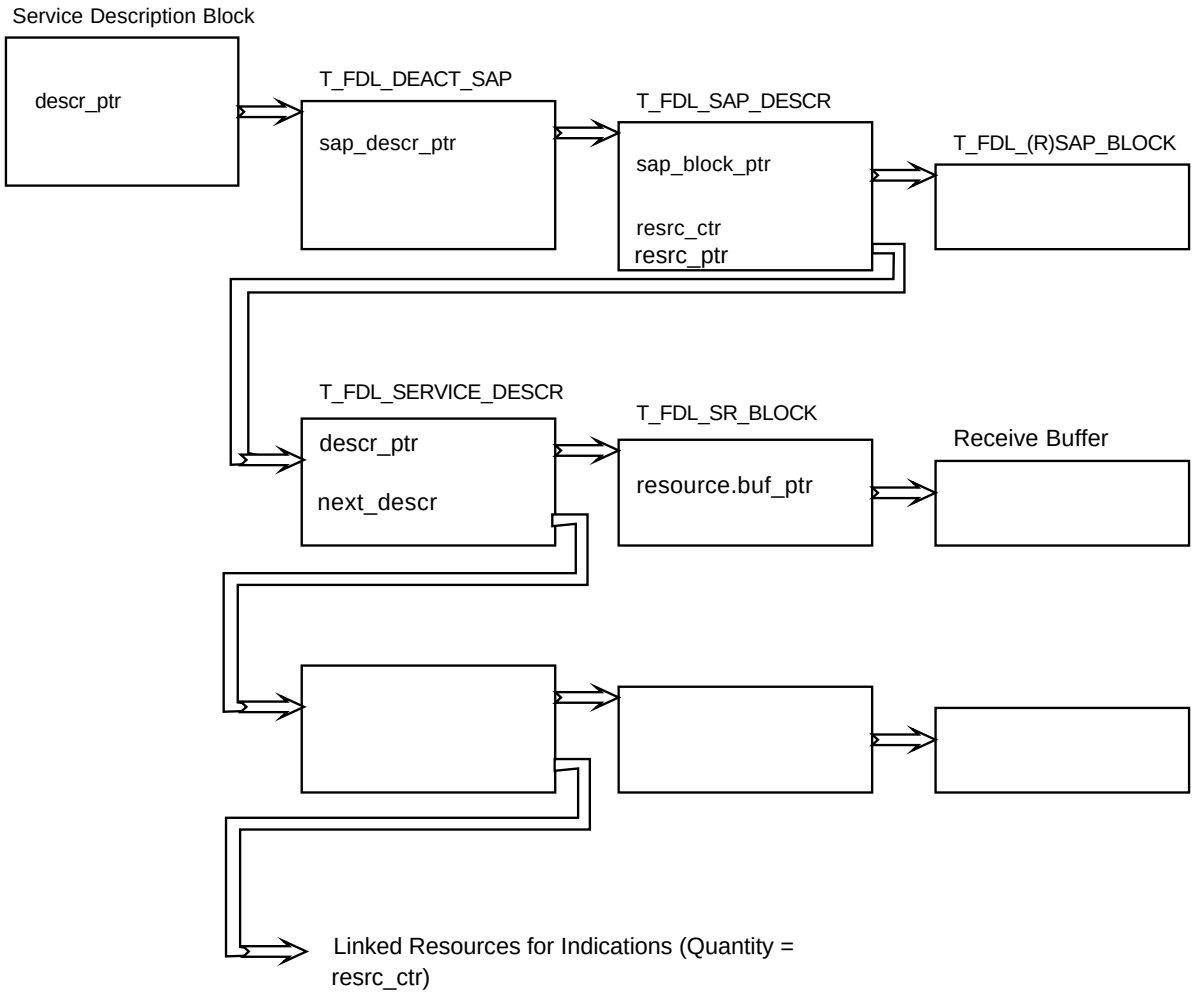
 © 1995 PEP Modular Computers

FMA2_DEACTIVATE_SAP Confirmation

Description:

The FDL confirms the deactivation of the Service Access Point and returns all data structures that were assigned to this Service Access Point, or an error status is flagged.

Data Structure:



The SAP deactivation block (*T_FDL_DEACT_SAP*) contains a pointer to the SAP description block, which as during the SAP activation points to the SAP block (*T_FDL_SAP_BLOCK* or *T_FDL_RSAP_BLOCK*).

Additionally, the parameter *resrc_ptr* in the SAP description block gives details about the "chain" of linked resources needed to process the indications assigned to this Service Access Point, and given to the FDL through the use of the *PUT_RESRC_TO_FDL* service.

 © 1995 PEP Modular Computers

Service Description Block:

sap	*remains unchanged*	
service	FMA2_DEACTIVATE_SAP	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, NO or IV	*see below*
descr_ptr	*remains unchanged*	Pointer to SAP deactivation block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

SAP Deactivation Block (T_FDL_DEACT_SAP)

ssap	*remains unchanged*	To deactivate the Service Access Point
sap_descr_ptr	(T_FDL_SAP_DESCR far *)	Pointer to SAP description block

Status Values:

Code	Meaning
OK	The Service Access Point is deactivated
NO	The desired Service Access Point does not exist
IV	Invalid parameter in request

 © 1995 PEP Modular Computers

4.5.8 Services for the Administration of the Resources

The layer 2 software requires a receiver buffer, into which the incoming telegrams are stored together with their respective parameter blocks, where also the parameters for the indications may be entered.

The FDL-User is therefore responsible in ensuring that these resources are always available (in layer 2) in sufficient forms. For the handover of resources to the FDL and to take back resources no longer required, three services are provided. These three services are described in the following pages.

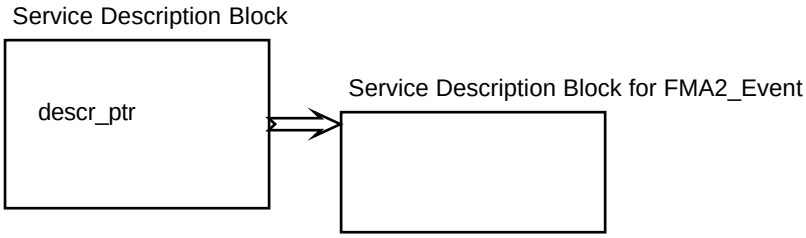
 © 1995 PEP Modular Computers

WAIT_FOR_FMA2_EVENT Request

Description:

The FDL-User supplies a resource to the FDL to process *FMA2_EVENTs*.

Data Structure:



The first Service Description Block is returned for the confirmation. The second remains in the FDL for the processing of the *FMA2_EVENTs*. Only one resource can be given to the FDL at a time.
The *FMA2_EVENTs* are stored in the FDL in a ring buffer. After receipt of a resource the oldest *FMA2_EVENT* is announced.

Service Description Block:

sap	*not significant*	
service	WAIT_FOR_FMA2_EVENT	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_SERVICE_DESCR far *)	Pointer to Service Description Block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Handed Over Service Description Block:

sap	*not significant*	
service	*not significant*	
primitive	*not significant*	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	*not significant*	
next_descr	*reserved for FDL*	

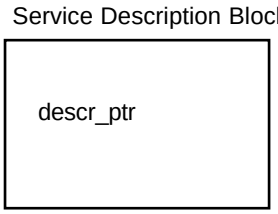
 © 1995 PEP Modular Computers

WAIT_FOR_FMA2_EVENT Confirmation

Description:

The provision of a resource for *FMA2_EVENTs* processing is confirmed, or an error status flagged.

Data Structure:



If status is "OK" only the Service Description Block for the *WAIT_FOR_FMA2_EVENT* services is returned. The hand-over Service Description Block remains in the FDL.

Service Description Block:

sap	*remains unchanged*	
service	WAIT_FOR_FMA2_EVENT	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, LR or IV	*see below*
descr_ptr	NULL	In the event of error remains unchanged
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Status Values:

Code	Meaning
OK	Resource accepted
LR	Resource not accepted, as the FMA2 already contains a resource
IV	Invalid parameter in request

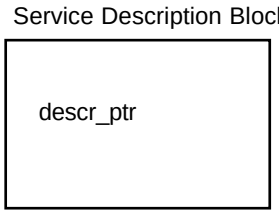
 © 1995 PEP Modular Computers

WITHDRAW_EVENT Request

Description:

The FDL-User withdraws the resource from the FDL to process *FMA2_EVENTs*.

Data Structure:



Service Description Block:

sap	*not significant*	
service	WITHDRAW_EVENT	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	*not significant*	
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

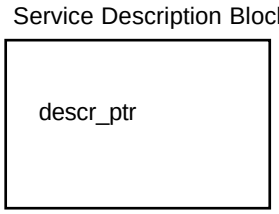
 © 1995 PEP Modular Computers

WITHDRAW_EVENT Confirmation

Description:

The resource for *FMA2_EVENTs* processing is confirmed, or an error status flagged.

Data Structure:



Service Description Block:

sap	*remains unchanged*	
service	WITHDRAW_EVENT	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, LR or IV	Status
descr_ptr	(T_FDL_SERVICE_DESCR far *)	Pointer to resource for *FMA2_EVENT*
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Status Values:

Code	Meaning
OK	Resource accepted
LR	Resource not available
IV	Invalid parameter in request

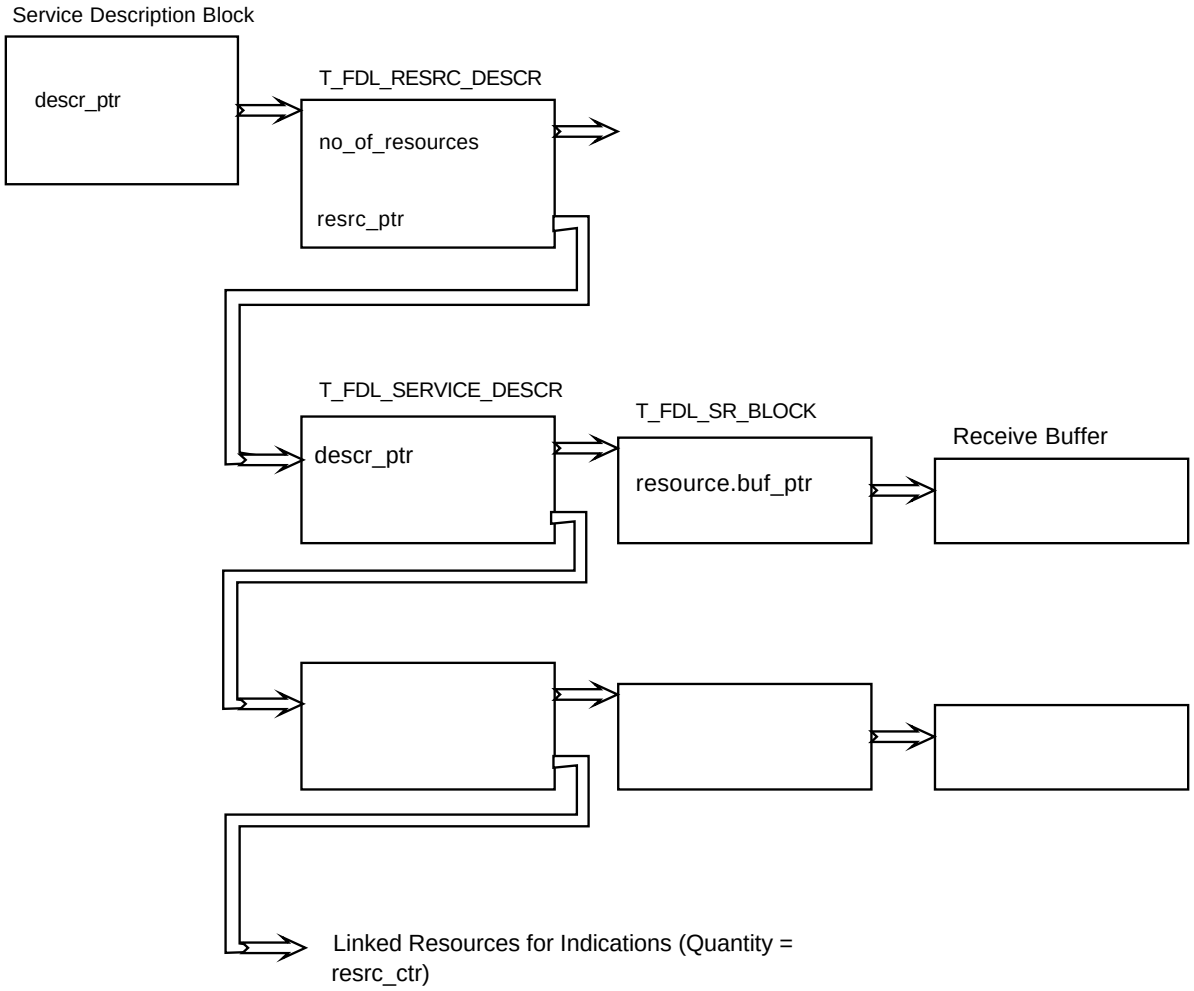
 © 1995 PEP Modular Computers

PUT_RESRC_TO_FDL Request

Description:

The FDL-User returns the FDL resources to process SDA, SDN and SRD indications (of a certain Service Access Point) or to process a CSRD confirmation of a defined Poll-List entry.

Data Structure:



Service Description Block:

sap	0..63 or DEFAULT_SAP	SAP to which the resources are assigned
service	PUT_RESRC_TO_FDL	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_RESRC_DESCR far *)	Pointer to resources descriptor
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

Resources Descriptor (T_FDL_RESRC_DESCR)

dsap	0..62 or DEFAULT_SAP	Service Access Point and
rem_add.station	0..126	partner address to identify the poll
rem_add.segment	0..62 or NO_SEGMENT	list entry in the Poll-List SAP
nr_of_resources	> 0	Quantity of the supplied resources
resrc_ptr	(T_FDL_SERVICE_DESCR far *)	Pointer to resource list

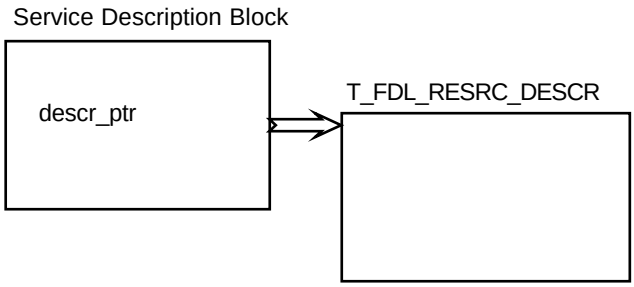
 © 1995 PEP Modular Computers

PUT_RESRC_TO_FDL Confirmation

Description:

The FDL confirms the acceptance of the resources for the required Service Access Point or Poll-List entry, or an error status is returned.

Data Structure:



If status is "OK" only the Service Description Block and the resource description block is returned; in the event of an error occurring the complete structure supplied during the request is returned.
The resource description block, therefore, only needs to be provided once and can be used many times again for the transfer of resources as required.

Service Description Block:

sap	*remains unchanged*	
service	PUT_RESRC_TO_FDL	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, NO or IV	*see below*
descr_ptr	(T_FDL_RESRC_DESCR far *)	Pointer to resource description block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Resources Description (T_FDL_RESRC_DESCR)

dsap	*remains unchanged*	
rem_add.station	*remains unchanged*	
rem_add.segment	*remains unchanged*	
nr_of_resources	*remains unchanged*	
resrc_ptr	Null	In the event of error remains unchanged

Status Values:

Code	Meaning
OK	The resources are accepted
NO	The desired Service Access Point or Poll-List does not exist
IV	Invalid parameter in request

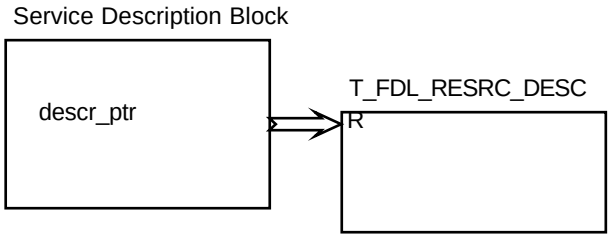
 © 1995 PEP Modular Computers

WITHDRAW_RESRC_FROM_FDL Request

Description:

The resources supplied for a particular Service Access Point or Poll-List entry are to be withdrawn.

Data Structure:



Service Description Block:

sap	0..63 or DEFAULT_SAP	SAP from which the resources are withdrawn
service	WITHDRAW_RESRC_FROM_FDL	
primitive	REQ	
user_id	0..65535	Identification possibility for FDL-User
status	*not significant*	
descr_ptr	(T_FDL_RESRC_DESCR far *)	Pointer to resource description block
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

Resource Description (T_FDL_RESRC_DESCR)

dsap	0..62 or DEFAULT_SAP	Service Access Point and
rem_add.station	0..126	partner address to identify the Poll-
rem_add.segment	0..62 or NO_SEGMENT	List entry in Poll-List SAP
nr_of_resources	*not significant*	
resrc_ptr	*not significant*	

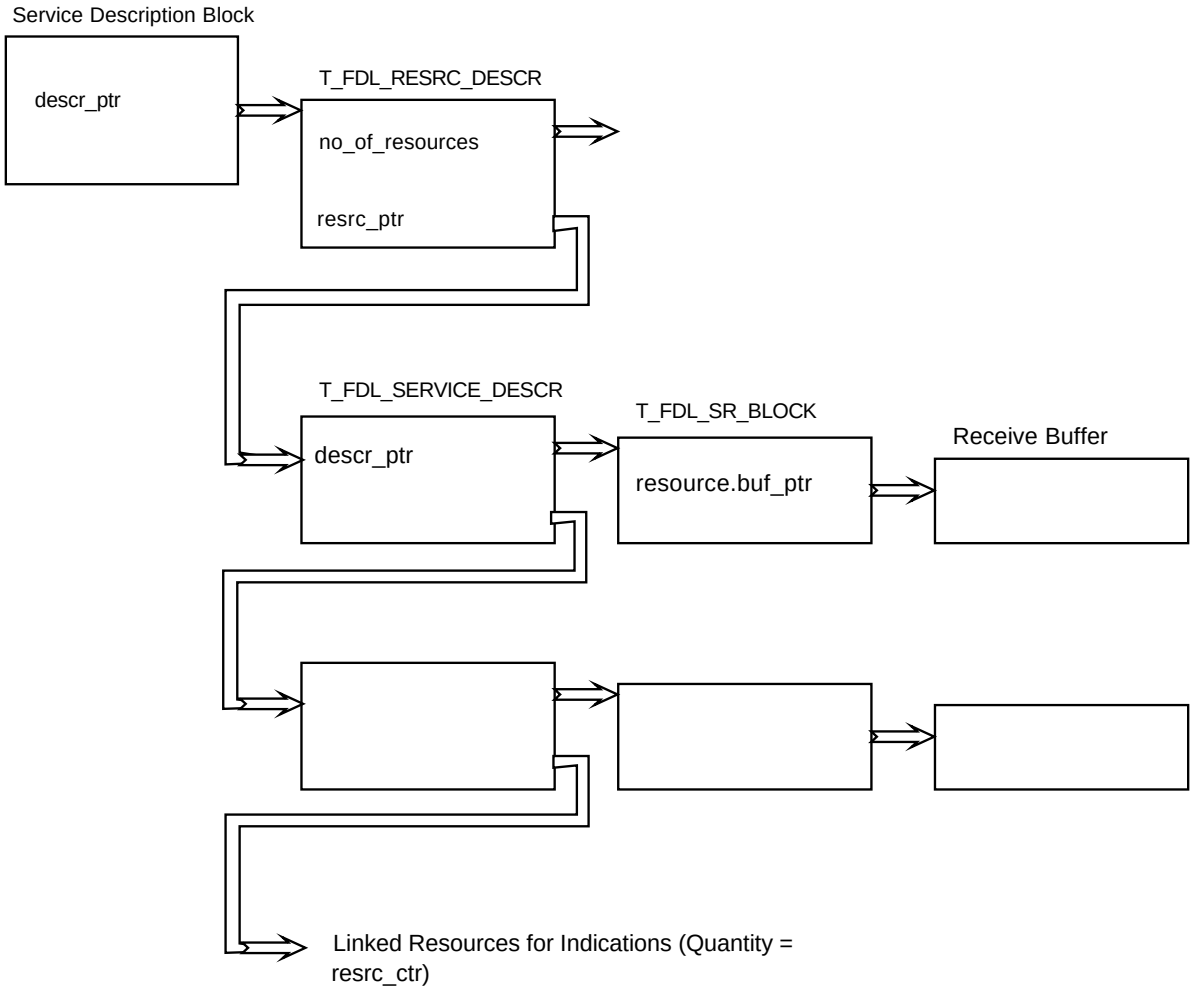
 © 1995 PEP Modular Computers

WITHDRAW_RESRC_FROM_FDL Confirmation

Description:

The FDL gives the resources from the designated Service Access Point or Poll-List entry back or flags an error status.

Data Structure:



Service Description Block:

sap	*remains unchanged*	
service	WITHDRAW_RESRC_FROM_FDL	
primitive	CON	
user_id	*remains unchanged*	Identification possibility for FDL-User
status	OK, LR or IV	Status
descr_ptr	*remains unchanged*	Pointer to resources description
next_descr	*reserved for FDL*	
link_descr	*reserved for FDL*	
resrv	*reserved for FDL*	

 © 1995 PEP Modular Computers

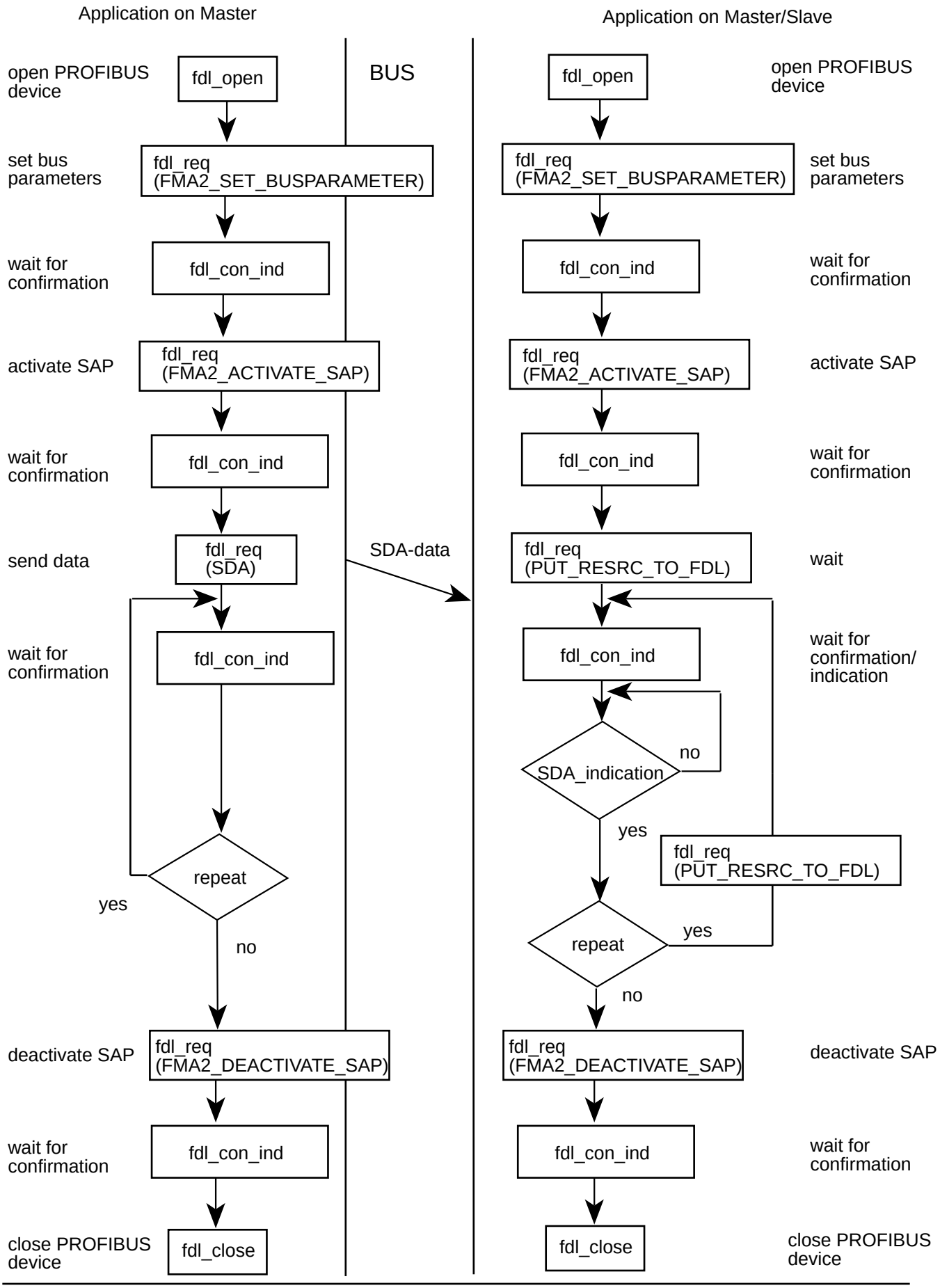
Resources Description (T_FDL_RESRC_DESCR)

dsap	*remains unchanged*	
rem_add.station	*remains unchanged*	
rem_add.segment	*remains unchanged*	
nr_of_resources	>= 0	Number of returned resources
resrc_ptr	(T_FDL_SERVICE_DESCR far *)	Pointer to resource list

Status Values:

Code	Meaning
OK	The resources are accepted
LR	The desired Service Access Point or Poll-List entry does not exist
IV	Invalid parameter in request

An application program structure using the `pbl2llf.l` library is shown overleaf.



 © 1995 PEP Modular Computers

4.5.9 Parameterizing Layer 2

The user is allowed to modify several entries in the following files in order to adapt them to his application requirements:

```
/PROFINET/BSP/VIUC/PBL2DESC/pSMART_<n>.a      (n = 1,2,3)
/PROFINET/BSP/VIUC/PBL2DESC/pVIUC_<n>.a
/PROFINET/BSP/VM30/PBL2DESC/pVM30_<n>.a
/PROFINET/BSP/COMMON/NFMDESC/n1PROFI.a
/PROFINET/BSP/COMMON/DATMOD/busPB.a
```

Additionally, the utility *pbmode* provides the possibility to modify the bus parameter setting in the *busPB* data module or during a running PROFIBUS application.

pSMART_<n>.a
pVIUC_<n>.a
pVM30_<n>.a

The user is allowed to change the following entries:

`D_CheckReq` - This entry is the flag to check send data service requests, such as SDA, SDN or SDR.

0x00	No parameter check in request services recommended when only Level 7 applications are running, as Layer 7 also checks the parameters
0x01 - 0xFF	Parameter check recommended when Layer 2 applications are running

`D_UseL7` - Flag to use L2 or L7 queue handling.

PEOFIBUS offers the user two levels of priority (LOW/HIGH) in order to deal with services. In the normal running of the application, low prioity message cycles are dealt with. If an alarm status is transmitted, high priority message cycles can be employed. High priority message cycles are dealt with first and , from a time viewpoint, can overtake low priority message cycles.

If Layer 2 is driven together with Layer 7, certain limitations must be taken into account when using high priority message cycles. According to the PROFIBUS standards, Layer 7 tasks can only be carried out when a connection to a station completed. The connection is always made before priority tasks. If a high priority message is generated immediately after the building of the connection, it could be that this message is dealt with before the connection acknowledgement. In this case, the connection is immediately cut off.

In order to deal with this problem, D_UseL7 must be set. The high and low priority queues can then be dealt with in such a way that no limitations of the choice of priority need occur under Layer 7.

0x00	L2 queue handling, high and low priority request service support recommended when Layer 2 applications are running
0x01 - 0xFF	L7 queue handling, high priority request handling recommended in the same way as low priority requests when only Layer 7 applications are running

`D_UseNLT` - Flag to support NLT error handling.

0x00	No NLT handling recommended
0x01 - 0xFF	NLT error handling. The status NLT is signalled when a remote call cannot be generated due to the fact that the active station does not join or was removed from the logical token ring.

`D_CountNLT` - Counter for NLT. This is only used if D_UseNLT <> 0.

 © 1995 PEP Modular Computers

n1PROFI.a

The following entry can be modified:

```
PB_SAP      equ  60
```

This is the selected PROFIBUS Service Access Point that is used for the OS-9/NET communication. It is recommended to modify the value for `PB_SAP` only if this SAP must be used in a PROFIBUS application.

busPB.a

This file is a pure data file and contains all bus parameters that are related to the *FMA2_SET_BUSPARAMETER* service. From this assembler source file `makefile` shows various object files that are stored under different data names. The module name is `busPB` for all the data.

Function

`open_PROFI` in the `pbL2hlf.l` library selects the bus parameters from the data module `busPB`, and initiates the service *FMA2_SET_BUSPARAMETER*.

This data module must be present in the OS-9 module directory if the function commands of the `pbL2hlf.l` library are required when OS-9/PROFINET is initiated or an application program is running. With a romable OS-9 the module `busPB` can be present in the EPROM.

`makefile` generates various object files from the base file `busPB.a`, whereby the following three different parameters are pre-set when assembling is complete:

`LOC_ADDR`	determines the bus parameter `station`
`MODE`	defines the bus parameter `in_ring_desired`
`PROFI`	selects the PROFIBUS devices */profi_ 1* or */profi_2*

The following files are generated with `makefile`:

`bPB1_1 - bPB1_10 stations`	10 data modules for PROFIBUS stations numbered 1-10; the device *profi_1* and thus the interface MC68302 SCC#1 is defined as a PROFIBUS connection.
`bPB2_1 - bPB2_10`	As above except using PROFIBUS device */profi_2*.
`bPB1_S`	DIP switch settings on the CXM or STAT-1 or STAT-2 are read and used to define the local PROFIBUS station address. The PROFIBUS device is */profi_1*.
`bPB2_S`	As `bPB1_S` except that the PROFIBUS device is */profi_2*.
`bPB1_I`	DIP switch settings on the IUC board defines the PROFIBUS station address. The PROFIBUS device is */profi_1*.
`bPB2_I`	As above except using PROFIBUS device */profi_2*.
`bPB1_M`	A pre-defined value in EEPROM on the SMART-I/O defines the PROFIBUS station address. The PROFIBUS device is */profi_1*.

 © 1995 PEP Modular Computers

The user has the capability to influence the setting of the PROFIBUS bus parameters (eg. baud rate) by changing the values in the data module busPB. The contents of the file busPB are in the following order:

```
BP_BLOCK.station          dc.b   PB_station
BP_BLOCK.station_mask     dc.b   PB_station_mask
BP_BLOCK.segment          dc.b   PB_segment
BP_BLOCK.baud_rate        dc.b   PB_baud_rate
BP_BLOCK.medium_red       dc.b   PB_medium_red
BP_BLOCK.tsl              dc.w   PB_tsl
BP_BLOCK.min_tsdr         dc.w   PB_min_tsdr
BP_BLOCK.max_tsdr         dc.w   PB_max_tsdr
BP_BLOCK.tqui             dc.b   PB_tqui
BP_BLOCK.ttr              dc.l   PB_ttr
BP_BLOCK.g                dc.b   PB_g
BP_BLOCK.in_ring_desired  dc.b   PB_in_ring_desired
BP_BLOCK.hsa              dc.b   PB_hsa
BP_BLOCK.max_retry_limit  dc.b   PB_max_retry_limit
BP_BLOCK.token_hold       dc.b   PB_token_hold
BP_BLOCK.ident            dc.l   PB_ident
BP_BLOCK.device           dc.l   PB_device
```

PB.station	This value is defined by the LOC_ADDR parameter after initiating the Assembler with -a=LOC_ADDR=<n>, the values for <n> can be: 0,1,2,...126, 128, 129, 130
0..126	Local PROFIBUS station address
128	The local station address is determined by one of the DIP switch settings of the status boards, CXM-STAT1 or -STAT2. This value is linked with PB_station_mask.
129	The local station address is defined by the DIP switch settings on the IUC board. This definition can only be selected if the PROFIBUS is booted with an IUC board fitted with on-board DIP switches. The value read is linked to PB_station_mask.
130	The local station address is defined by a value stored in EEPROM. This value can only be selected if PROFIBUS is running on a SMART I/O board.

Note

Don't select station address = 0 if you are running OS-9/NET on PROFIBUS in parallel.

 © 1995 PEP Modular Computers

PB_station_mask

This is only used when PB.station>127.
Bit 7 is only controlled by parameter MODE
MODE is not specified or
MODE = 0 -> bit 7 = 0
MODE = 1 -> bit 7 = 1

Bit 6 depends on the value of bit 7.
If bit 7 = 0:
mask bit for corresponding DIP-switch to define the station address.
The station mode depends on the chosen value of PB_in_ring_desired (default = 0xFF, active).
If bit 7 = 1:
value of corresponding DIP switch defines the value for PB_in_ring_desired and therefore the station mode active or passive.

Bit 5 - bit 0: mask bits for corresponding DIPnswitches to define the station address

Examples:
PB_station_mask = 0x8F
bit 7 = 1 -> DIP switch 6 defines station mode
bit 4 - bit 0 = 1 -> DIP switches 0 - 4 determine the station address

PB_station_mask = 0x7F
bit 7 = 0 and bit 0 - 6 = 1 -> DIP switches 0 - 6 determine the station address

PB_segment

Local segment address
0...63 or 255 (NO_SEGMENT)
Default: NO_SEGMENT

PB_baud_rate

Baud rate, valid baud rate codes are:
0 = 9600 (K_BAUD_9_6)
1 = 19200 (K_BAUD_19_2)
2 = 93750 (K_BAUD_93_75)
3 = 187500 (K_BAUD_187_5)
4 = 500000 (K_BAUD_500)
Default: K_BAUD_187_5
Depending on the selected baud rate code following parameters in busPB.a are preset with recommended values:
PB_tsl
PB_min_tsdr
PB_max_tsdr
PB_tqui
PB_tset
PB_ttr
PB_g
The values of these parameters with the exception of PB_g are calculated in bit times. PB_g is a multiple factor of PB_ttr.

PB_medium_red

Valid values:
0: NO_REDUNDANCY
1: REDUNDANCY
Default: NO_REDUNDANCY

 © 1995 PEP Modular Computers

PB_tsl	Slot time: 1...65535 Default: 3500
PB_min_tsdr	Minimum station delay time: 1...65535 Default: 500
PB_max_tsdr	Maximum station delay time: 1...65535 Default: 1000
PB_tqui	Modulator decouple time: 0...255 Default: 0
PB_tset	Set up time: 1...255 Default: 50
PB_ttr	Target rotation time: 1...0xFFFFFF Default: 100000
PB_g	GAP update factor: 1...100 Default: 1
PB_in_ring_desired	Desired role of station (passive/active) 0: passive station (FALSE) 0xFF: active station (TRUE) The value of PB_in_ring_desired depends on the r68 parameter MODE and LOC_ADDR.
PB_hsa	Highest station address: 2...126 Default: 20
PB_max_retry_limit	Maximum telegram retries: 1...8 Default: 1
PB_token_hold	Token hold time (bit time) 0: No token hold 1...255: Token hold time A token hold time is only recommended, if there are not more than two active stations connected to the logical token ring. Default: 0
PB_ident	Offset pointer to the identity field
PB_device	Offset pointer to the PROFIBUS device name. The PROFIBUS device name depends on the value of the r68 parameter PROFI. Possible names are: /profi_1 /profi_2 /profi_3

 © 1995 PEP Modular Computers

pbmode

The *pbmode* utility provides the possibility to modify the bus parameters defined in the *busPB* data module. Normally the modifications are only effective when the changes are made before a PROFIBUS application has been run. Bus parameter changes can be made dynamically during the running of a PROFIBUS application using the option '-c'.

Function

Modify the bus parameters.

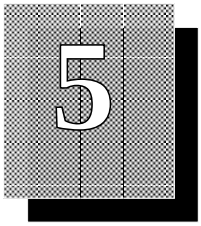
Syntax

```
pbmode <options>
```

Options

Option	Parameter	Value
-p	Default parameter dependent on the baud rate	
-c	Change parameter dynamically on Layer 2	
-h=	Highest station address	2 .. 126
-m=	In ring desired	0 (FALSE), <> 0 (TRUE)
-a=	Station address	0 .. 126, 128, 129, 130
	0 .. 126: define station address	
	128: CxM-STATx define station address by DIP switch	
	129: IUC define station address by DIP switch	
	130: SMART I/O define station address by DIP switch	
-w	Write station address	0 .. 126
	Valid only if modification is possible	
-v=	Station mask	0X00, 0xFF
-s=	Station segment	0 .. 63/255
-b=	Baud rate	0, 1, 2, 3, 4
-g=	Gap update	1 .. 100
-r	Maximal retry limit	1 .. 8
-d=	Token delay	0 (no delay), 1 .. 255
-tt=	Target rotation time (*trr*)	1 .. 16777215 (0xFFFFFF)
-ts	Slot time (*tsl*)	1 .. 65535 (0xFFFF)
-??=	Minimum station delay time (*min_tsdr*)	1 .. 65535 (0xFFFF)
-??=	Maximum station delay time (*max_tsdr*)	1 .. 65535)0xFFFF)
-tq=	Modulator decouple time (*tqui*)	0 .. 255 (0xFF)
-te=	Exposure time (*tset*)	1 .. 255 (0xFF)

 © 1995 PEP Modular Computers



5. RELEASE NOTES

OS9/PROFINET - Edition History

21/12/92:

OS9/PROFINET V3.1 first release.

Current edition numbers of PROFIBUS modules:

`phyPROFI`	edition #9
`drvPROFI`	edition #4
`profiman`	edition #6
`nfPROFI`	edition #1
`comPROFI`	edition #5

12/02/93:

OS9/PROFINET V3.1/I1.1.

New:

PROFIBUS Layer 7 is now supported.

Major changes:

- The structure of *T_FDL_SEVICE_DESCR* defined in `pbL2type.h` has been extended, all PROFIBUS Layer 2 applications must be recompiled with the updated `pbL2type.h` file.

- There is a new file structure:
  All OS-9/NET specific files are now included under the directory `NET`/ (e.g. nfm, chp ...). The others remain under the PROFINET/ directory.

Current edition numbers of PROFIBUS modules:

`phyPROFI`	edition #10
`drvPROFI`	edition #5
`profiman`	edition #7
`nfPROFI`	edition #1
`comPROFI`	edition #6
`modPBL7`	edition #3
`srvPBL7`	edition #1

 © 1995 PEP Modular Computers

Note: PROFIBUS applications compiled with *pbL2type.h* and/or using library *pbL2hlf.l* from OS-9/PROFINET V3.1 do not run with the new PROFIBUS modules. If this is the case, recompile with updated files.

The behaviour of PROFIBUS Layer 2 service *FMA2_SET_BUSPARAMETER* has been changed, refer to example program `sda_demo.c`.

Do not mix PROFIBUS modules from different releases.

VIUC:

VIUC-board equipped with a PEPbug Monitor (up to Version 568-3).

If the upper SCC port of the VIUC is used as the PROFIBUS interface, the initialization of that port by the PEPbug Monitor disturbs the PROFIBUS protocol of the remaining PROFIBUS stations that are connected to the network as long as the VIUC upper port is not re-initialized by the PROFIBUS driver.

22/06/93:

OS9/PROFINET V3.1/I1.2.

Major changes:

- Modifications in module `phyPROFI`:
  Task using PROFIBUS services could hang in system call for an event (occurred only when a large amount of noise was present on the PROFIBUS cable).
  Support of 500 KBaud.

- Modifications in module `profiman`:
  User signals are now returned.

- Modifications in the PROFIBUS device descriptors and "*busPB.a*" file are now allowed to run 500 KBaud (the PROFIBUS board must be equipped with a MC68302-20MHz and a 24MHz oscillator).

- File names of "*busPB*" modules, generated from the "*busPB.a*" file, adapted to MS-DOS file name conventions (e.g. *busPB1_1* now *bPB1_1*).

- Library `pbL2hlf.l` extended:
  Several functions added.
  Modified behaviour of SRD Indications:
  Responser SAP initialized with indication_mode == ALL (instead of indication_mode == DATA).

- Application "*pbmode*" added to modify bus parameters in the "*busPB*" data module. This affects only the PROFIBUS initialization. If the modification is performed before initialization, a task opens the PROFIBUS and the task must use the function call "*open_PROFI*" of the *pbL2hlf.l* library for opening.

Current edition numbers of PROFIBUS modules:

`phyPROFI`	edition #14
`drvPROFI`	edition #5
`profiman`	edition #8
`nfPROFI`	edition #1
`comPROFI`	edition #6
`modPBL7`	edition #3
`srvPBL7`	edition #1

 © 1995 PEP Modular Computers

20/01/94:

OS9/PROFINET V3.1/I1.3

PROFIBUS V3.1/I1.3 on a VM30 or a (V)IUC can be used with the OS-9 Professional V2.4/I2.2.
OS-9/RAMNET V1.8 must be installed when using PROFIBUS V3.1/I1.3 on a VIUC as a VMEbus Slave.

Current edition numbers of PROFIBUS modules:

`phyPROFI`	edition #15
`drvPROFI`	edition #5
`profiman`	edition #8
`nfPROFI`	edition #2
`comPROFI`	edition #8
`modPBL7`	edition #4
`srvPBL7`	edition #2

Major changes:

- The PROFIBUS implementation runs now also on a VM30 with a MC68030-CPU and installed OS9-module "ssm". The following modules/files have now been changed:
  `nfPROFI`
  `comPROFI`
  `srvPBL7`
  `modPBL7`
  `pbL7llf.l`

- Modifications in module *phyPROFI* (ed #15):
  Automatic recognition of CPU frequency for MC68302 and external frequency on TIN1-PIN of MC68302
  -> one PROFIBUS device descriptor for a VM30 or (V)IUC supports both variants:

MC68302 on VM30:	CPU frequency	External frequency
	16 MHz	12 MHz
	20 MHz	24 MHz
MC68302 on (V)IUC:	CPU frequency	External frequency
	16.67 MHz	12 MHz
	20 MHz	24 MHz

The entries "*D_CPUFreq*" and "*D_EXTFreq*" in the PROFIBUS device descriptor are ignored by the PROFIBUS driver.

- Modifications in module *busPB* (ed #2):
  Values for bus parameters are changed according to the recommandations of the PNO

- Bug fixes in library *pbL2hlf.l* :
  function "*open_PROFI*" returns now zero instead of the PROFIBUS station number, if no error occurred
  function "*close_PROFI*" returns zero if no error occurred
  -> the Layer 2 application examples are re-linked with the updated library

- Modification in file /PROFINET/ROM/VIUC/makefile:
  "fpu" module is now included

 © 1995 PEP Modular Computers

20/07/94:

OS9/PROFINET V3.1/I1.4

PROFIBUS V3.1/I1.4 on a VM30 or a (V)IUC can be used with the OS-9 Professional V2.4/I2.2.
The OS-9/RAMNET V1.8 must be installed when using PROFIBUS V3.1/I1.4 on a VIUC as a VMEbus Slave.

Current edition numbers of PROFIBUS modules:

`phyPROFI`	edition #16
`drvPROFI`	edition #5
`profiman`	edition #8
`nfPROFI`	edition #2
`comPROFI`	edition #8
`modPBL7`	edition #4
`srvPBL7`	edition #3

Major changes:

- Bug fix in library *pbL7llf.l*
- Application examples changes to Ultra-C notation

20/11/94:

OS9/PROFINET V3.12/I1.0

PROFIBUS V3.12/I1.0 on VM30/(V)IUC or SMART-I/O can be used with OS-9 Professional V3.0.

Current edition numbers of PROFIBUS modules:

`phyPROFI`	edition #17
`drvPROFI`	edition #6
`profiman`	edition #9
`nfPROFI`	edition #2
`comPROFI`	edition #9
`modPBL7`	edition #5
`srvPBL7`	edition #3

Major changes:

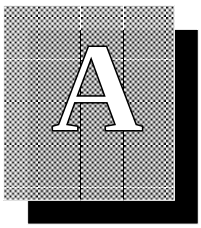
- Update of Profibus Layer 2 and 7 to V3.12
- Support of SMART-I/O

Note:

- Due to changes in Layer 2 and Layer 7 definition files and in libraries *pbL2hlf.l* and *pbL7llf.l*, applications must be re-compiled. Modifications in source code are possibly necessary, because structure definitions for Layer 7 services have been changed.

- Important modification in Layer 2:
  use GLOBAL_ADDR instead of ALL, when all stations should be accessed.

 © 1995 PEP Modular Computers



APPENDIX A STATUS VALUES

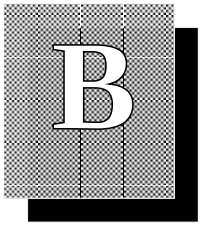
The following list provides the "general" meanings of the status values. Where localization occurs, the deviations are described in their respective service description sections.

Name	Code	Description
OK:	00	Positive confirmation that service(s) is carried out
UE:	01	Interface error
RR:	02	The partner did not have adequate operational resources
RS:	03	Partners service, access authorization or SAP, is not activated
HI:	05	update_status for SRD.ind: High priority reply data has been collected
LO:	06	update_status for SRD.ind: Low priority reply data has been collected
		update_status for CSRD.con: Data has been transfered in reply telegram
DL:	08	For SRD and CSRD services:Reply Data low available, positive confirmation of data sent
NR:	09	For SRD and CSRD services: No Reply Data available, positive confirmation of data sent
DH:	10	For SRD and CSRD services: Reply Data high available, positive confirmation of data sent
RDL:	12	For SRD and CSRD services: Reply Data low available, negative confirmation of data sent
RDH:	13	For SRD and CSRD services: Reply Data high available, negative confirmation of data sent
LS:	16	Service or local Service Access Point not activated
NA:	17	Addressed partner does not respond
NLT:	18	Own station not in logical token ring or has left bus
NO:	19	update_status for SRD.ind: No reply data is transfered
		update_status for CSRD.con: No reply data is transfered in reply telegram
LR:	20	No or insufficient operational resources are available locally
IV:	21	Invalid parameter in request

 © 1995 PEP Modular Computers

This page has been intentionally left blank

 © 1995 PEP Modular Computers



APPENDIX B DEFINITION OF CONSTANTS

The include data "pbL2con.h" contains all constants needed to call up layer 2 software.

The meanings of the constants are as described by their respective services.

```
/*****************************************************************************
*                                                                            *
*                        Include File pbL2con.h                              *
*                                                                            *
*                   *
* The include file pbL2con.h contains all constants that the user needs      *
* for programming LAYER2.      *
*                    *
*****************************************************************************/



/*****************************************************************************
*                                                                            *
* Edition History                                                            *
* ===============                                                            *
*                                                                            *
* #    Date   Comments                                              by    *
* -- -------- ------------------------------------------------------ ----  *
* 01 11/03/92 First written                                         HAH    *
*                                                                            *
* 02 01/27/93 Added new service definition:                         HAH    *
*                #define FMA2_CHANGE_BUPARAMETER   0x1F*
*                                                                            *
* 03 12/23/93 Added ident definition:  HWE    *
*               #define _PBL2CON_HEADER_ and surrounding #ifndef    *
*                                                                            *
*****************************************************************************/

#ifndef _PBL2CON_HEADER_/* include it only once */
#define _PBL2CON_HEADER_
```

 © 1995 PEP Modular Computers

```
/* ---------------------------------------------------------------------------
+                   +
+        Definition of Boolean Constants      +
+                   +
+ ------------------------------------------------------------------------- */

#define TRUE    -1
#define FALSE   0



/* ---------------------------------------------------------------------------
+                   +
+        Definition of the Service Primitives     +
+                   +
+ ------------------------------------------------------------------------- */

#define REQ     0
#define CON     1
#define IND     2



/* ---------------------------------------------------------------------------
+                   +
+        Definition of Constants for the Length of FDL Telegram Header and    +
+        FDL Telegram Trailer     +
+                   +
+ ------------------------------------------------------------------------- */

#define FDL_OFFSET11
#define FDL_TRAILER 2


/* ---------------------------------------------------------------------------
+                   +
+        Definition of Constants for the Length of IDENT Telegram and         +
+        LSAP Status Telegram         +
+                   +
+ ------------------------------------------------------------------------- */

#define IDENT_TELE_LEN255
#define LSAP_STATUS_TELE_LENFDL_OFFSET + FDL_TRAILER + 6
```

 © 1995 PEP Modular Computers

```
/* ---------------------------------------------------------------------------
+                   +
+        Definition of the Service Codes for FDL and FMA2 Services     +
+                   +
+ --------------------------------------------------------------------------- */

#define SDA            0x01
#define SDN           0x02
#define SRD           0x03
#define CSRD          0x04
#define LOAD_POLL_LIST0x05
#define DEACT_POLL_LIST0x06
#define POLL_ENTRY0x07
#define SEND_UPDATE0x08
#define REPLY_UPDATE0x09

#define FMA2_RESET      0x10
#define FMA2_SET_BUSPARAMETER  0x11
#define FMA2_SET_STATISTIC_CTR  0x12
#define FMA2_READ_BUSPARAMETER0x13
#define FMA2_READ_STATISTIC_CTR 0x14
#define FMA2_READ_TRR0x15
#define FMA2_READ_LAS0x16
#define FMA2_READ_GAPLIST0x17
#define FMA2_EVENT0x18
#define FMA2_IDENT0x19
#define FMA2_LSAP_STATUS0x1A
#define FMA2_LIVELIST0x1B
#define FMA2_ACTIVATE_SAP 0x1C
#define FMA2_ACTIVATE_RSAP 0x1D
#define FMA2_DEACTIVATE_SAP 0x1E
#define FMA2_CHANGE_BUSPARAMETER  0x1F

#define WAIT_FOR_FMA2_EVENT0x20
#define PUT_RESRC_TO_FDL0x21
#define WITHDRAW_RESRC_FROM_FDL0x22
#define WITHDRAW_EVENT0x23
```

 © 1995 PEP Modular Computers

```
/* ---------------------------------------------------------------------------
+                   +
+        Definition of Confirmation Status and Update Status of      +
+        SRD-Indications and CSRD-Confirmations.     +
+                   +
+ ------------------------------------------------------------------------- */

#define OK     0x00
#define UE     0x01
#define RR     0x02
#define RS     0x03
#define HI     0x05
#define LO     0x06
#define DL     0x08
#define NR     0x09
#define DH     0x0a
#define RDL    0x0c
#define RDH    0x0d
#define LS     0x10
#define NA     0x11
#define NLT    0x12/* corresponds to status DS in DIN 19245 Teil 1     */
#define NO     0x13
#define LR     0x14
#define IV     0x15



/* ---------------------------------------------------------------------------
+                   +
+        Definition of Broadcast SAP, Default SAP and FMA2 SAPs     +
+                   +
+ ------------------------------------------------------------------------- */

#define BRCT_SAP0x3F
#define DEFAULT_SAP 128

#define MSAP_0 0xF0
#define MSAP_1 0xF1
#define MSAP_2 0xF2


/* ---------------------------------------------------------------------------
+                   +
+        Definition of Constant NO_SEGMENT (for component 'segment' in     +
+        type T_FDL_ADDR)     +
+                   +
+ ------------------------------------------------------------------------- */

#define NO_SEGMENT0xFF
```

 © 1995 PEP Modular Computers

```
/* ------------------------------------------------------------------------
+                   +
+        Definition of Constants for services FMA2_SET_BUSPARAMETER and     +
+        FMA2_READ_BUSPARAMETER     +
+                   +
+ ------------------------------------------------------------------------ */

/* Baud rate ------------------------------------------------------------- */

#define K_BAUD_9_60
#define K_BAUD_19_21
#define K_BAUD_93_752
#define K_BAUD_187_53
#define K_BAUD_5004

/* Redundancy ------------------------------------------------------------ */

#define NO_REDUNDANCY 0
#define REDUNDANCY 1




/* ------------------------------------------------------------------------
+                   +
+        Definition of Constants for SAP Activation     +
+                   +
+ ------------------------------------------------------------------------ */

/* Service type ---------------------------------------------------------- */

#define SDA_RESERVED0x00
#define SDN_RESERVED0x01
#define SRD_RESERVED0x03
#define CSRD_RESERVED0x05

/* Role in Service ------------------------------------------------------- */

#define INITIATOR0x00
#define RESPONDER0x10
#define BOTH_ROLES0x20
#define SERVICE_NOT_ACTIVATED0x30



/* ------------------------------------------------------------------------
+                   +
+        Definition of Service Class in Send Requests and Indications     +
+                   +
+ ------------------------------------------------------------------------ */

#define LOW    0
#define HIGH   1
```

 © 1995 PEP Modular Computers

```
/* ---------------------------------------------------------------------------
+                   +
+       Definition of Transmit Mode in Send Update Requests and Reply      +
+       Update Requests          +
+                   +
+ ------------------------------------------------------------------------- */

#define SINGLE 0xF0
#define MULTIPLE0xF1


/* ---------------------------------------------------------------------------
+                   +
+       ALL is Global Address and Confirm Mode in Poll List      +
+       respectively Indication Mode in Responder SAP     +
+                   +
+       DATA is Confirm Mode in Poll List and Indication Mode      +
+       in Responder SAP     +
+                   +
+ ------------------------------------------------------------------------- */

#define ALL    0xFF
#define GLOBAL_ADDR0x7F

#define DATA   0xF0


/* ---------------------------------------------------------------------------
+                   +
+       Definition of Poll List Entry Marker State (Service POLL_ENTRY)     +
+                   +
+ ------------------------------------------------------------------------- */

#define UNLOCKED0x00
#define LOCKED 0x01

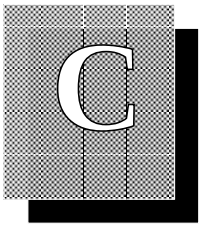

/* ---------------------------------------------------------------------------
+                   +
+       Definition FMA2 events     +
+                   +
+ ------------------------------------------------------------------------- */

#define FMA2_FAULT_ADDRESS0x01/* Duplicate address recognized*/
#define FMA2_FAULT_TRANSCEIVER0x02/* Transceiver error occured*/
#define FMA2_FAULT_TTO0x03/* Time out on BUS detected*/
#define FMA2_FAULT_SYN0x04/* No receiver synchronization*/
#define FMA2_FAULT_OUT_OF_RING0x05/* Station out of ring*/
#define FMA2_GAP_EVENT 0x06/* New station in ring*/


#endif /*_PBL2CON_HEADER_*/

/* ---------------------- END OF FILE ------------------------ */
```

 © 1995 PEP Modular Computers



APPENDIX C. TYPE DEFINITIONS

The definitions of the data types arriving at the communications interface are contained in the include data "pbL2type.h".

```
/*****************************************************************************
*                                                                            *
*                        Include File pbL2type.h                             *
*                                                                            *
*                    *
* The include file pbL2type.h contains all structures that the user needs    *
* for programming LAYER2.      *
*                     *
*****************************************************************************/


/*****************************************************************************
*                                                                            *
* Edition History                                                            *
* ===============                                                            *
*                                                                            *
* #    Date   Comments                                              by   *
* -- -------- ------------------------------------------------------ ---- *
* 01 11/03/92 First written                                        HAH  *
*              *
* 02 01/27/93 T_FDL_SERVICE_DESCR extended for:                    HAH  *
*              "profiman" edition #7                                   *
*              "phyPROFI" edition #10                                  *
*              *
* 03 02/14/94 Added ident. define:                                 HWE  *
*              #define _PBL2TYPE_HEADER_ and surrounding #ifndef*
*                Modified structure definition T_FDL_SERVICE_DESCR:          *
*               previous:*
*              USIGN8 far ** descr_ptr;*
*               now:*
*              void* descr_ptr;*
*              *
*                Modified structure definition T_FDL_SAP_DESCR:           *
*               previous:*
*              USIGN8 far *sap_block_ptr;*
*               now:*
*              voidfar * sap_block_ptr;*
*              *
*****************************************************************************/
```

 © 1995 PEP Modular Computers

```
#ifndef _PBL2TYPE_HEADER_ /* include if only once */
#define _PBL2TYPE_HEADER_

/* ------------------------------------------------------------------------ +
+  Definition of Base Types     +
+  ------------------------------------------------------------------------ */

#define far
#define VOID void
#define BOOL char
#define INT8 char
#define INT16 short
#define USIGN8 unsigned char
#define USIGN16 unsigned short
#define USIGN32 unsigned long

/* ------------------------------------------------------------------------ +
+  Definition of LAYER2 Types     +
+  ------------------------------------------------------------------------ */

typedef struct T_FDL_ADDR
{
USIGN8         station;
USIGN8         segment;
} T_FDL_ADDR;


typedef struct T_FDL_PDU
{
USIGN8 far *   buffer_ptr;
USIGN8         length;
} T_FDL_PDU;


typedef struct T_FDL_SERVICE_DESCR
{
USIGN8         sap;
USIGN8         service;
USIGN8         primitive;
USIGN8         path_id;/* reserved for OS-9 PROFIBUS Manager */
USIGN16        user_id;
USIGN8         status;
void           * descr_ptr;
struct T_FDL_SERVICE_DESCR far * next_descr;
struct T_FDL_SERVICE_DESCR far * link_descr;
               /* reserved for OS-9 PROFIBUS Manager */
USIGN32        resrv;/* reserved for OS-9 PROFIBUS Manager */
               /* currently not used                 */
} T_FDL_SERVICE_DESCR;
```

 © 1995 PEP Modular Computers

```
/* ------------------------------------------------------------------------ */

typedef struct T_BUSPAR_BLOCK
{
T_FDL_ADDR     loc_add;
USIGN8         baud_rate;
USIGN8         medium_red;
USIGN16        tsl;
USIGN16        min_tsdr;
USIGN16        max_tsdr;
USIGN8         tqui;
USIGN8         tset;
USIGN32        ttr;
USIGN8         g;
BOOL           in_ring_desired;
USIGN8         hsa;
USIGN8         max_retry_limit;
USIGN8 far *   ident;
USIGN8         ind_buf_len;
} T_BUSPAR_BLOCK;



/* ------------------------------------------------------------------------ */


typedef struct T_FDL_SAP_BLOCK
{
USIGN8         max_len_sda_req_low;
USIGN8         max_len_sda_req_high;
USIGN8         max_len_sda_ind_low;
USIGN8         max_len_sda_ind_high;
USIGN8         max_len_sdn_req_low;
USIGN8         max_len_sdn_req_high;
USIGN8         max_len_sdn_ind_low;
USIGN8         max_len_sdn_ind_high;
USIGN8         max_len_srd_req_low;
USIGN8         max_len_srd_req_high;
USIGN8         max_len_srd_con_low;
USIGN8         max_len_srd_con_high;
} T_FDL_SAP_BLOCK;
```

 © 1995 PEP Modular Computers

```
typedef struct T_FDL_SAP_DESCR
{
USIGN8          sap_nr;
T_FDL_ADDR      rem_add;
USIGN8          sda;
USIGN8          sdn;
USIGN8          srd;
USIGN8          csrd;
USIGN8          services;
void            far * sap_block_ptr;
T_FDL_SERVICE_DESCR far * resrc_tail;
T_FDL_SERVICE_DESCR far * resrc_hdr;
USIGN8          resrc_ctr;
USIGN8          sema;
} T_FDL_SAP_DESCR;


typedef struct T_FDL_RSAP_BLOCK
{
USIGN8          indication_mode;
USIGN8          max_len_upd_req_low;
USIGN8          max_len_upd_req_high;
USIGN8          max_len_srd_ind_low;
USIGN8          max_len_srd_ind_high;
USIGN8          max_len_sdn_ind_low;
USIGN8          max_len_sdn_ind_high;
T_FDL_PDU       upd_buf_low;
T_FDL_PDU       upd_buf_high;
T_FDL_PDU       telegram_low;
T_FDL_PDU       telegram_high;
USIGN8          transmit_low;
USIGN8          transmit_high;
USIGN8          marker_low;
USIGN8          marker_high;
USIGN8          fcs_low;
USIGN8          fcs_high;
} T_FDL_RSAP_BLOCK;


typedef struct T_FDL_DEACT_SAP
{
USIGN8          ssap;
T_FDL_SAP_DESCR far *sap_descr_ptr;
} T_FDL_DEACT_SAP;


/* ------------------------------------------------------------------------ */
```

 © 1995 PEP Modular Computers

```
typedef struct T_FDL_SR_BLOCK
{
T_FDL_ADDR     loc_add;
USIGN8         remote_sap;
T_FDL_ADDR     rem_add;
USIGN8         serv_class;
USIGN8         update_status;
T_FDL_PDU      send_data;
T_FDL_PDU      receive_data;
T_FDL_PDU      resource;
} T_FDL_SR_BLOCK;


typedef struct T_FDL_UPDATE_BLOCK
{
USIGN8         dsap;
T_FDL_ADDR     rem_add;
USIGN8         serv_class;
USIGN8         transmit;
T_FDL_PDU      upd_data;
} T_FDL_UPDATE_BLOCK;



/* ------------------------------------------------------------------------ */

typedef struct T_POLL_LIST_ELEMENT
{
USIGN8         dsap;
T_FDL_ADDR     rem_add;
USIGN8         max_len_csrd_req_low;
USIGN8         max_len_csrd_con_low;
USIGN8         max_len_csrd_con_high;
T_FDL_PDU      poll_buffer;
T_FDL_PDU      send_data;
T_FDL_SERVICE_DESCR far *resrc_hdr;
T_FDL_SERVICE_DESCR far *resrc_tail;
USIGN8         resrc_ctr;
T_FDL_SERVICE_DESCR far *next_for_rcv;
USIGN8         transmit;
USIGN8         to_send;
USIGN8         marker;
T_FDL_PDU      poll_telegram;
T_FDL_PDU      data_telegram;
USIGN8         data_fcs;
USIGN8         poll_fcs;
} T_POLL_LIST_ELEMENT;


typedef T_POLL_LIST_ELEMENT far * T_POLL_LIST_ELEM_PTR;
```

 © 1995 PEP Modular Computers

```
typedef struct T_POLL_LIST_DESCR
{
USIGN8          len;
USIGN8          confirm_mode;
T_POLL_LIST_ELEM_PTR far *elem_ptr;
} T_POLL_LIST_DESCR;


typedef struct T_POLL_ENTRY
{
USIGN8          dsap;
T_FDL_ADDR      rem_add;
USIGN8          marker;
} T_POLL_ENTRY;



/* ------------------------------------------------------------------------ */

typedef struct T_FDL_RESRC_DESCR
{
USIGN8          dsap;
T_FDL_ADDR      rem_add;
USIGN8          nr_of_resources;
T_FDL_SERVICE_DESCR far *resrc_ptr;
} T_FDL_RESRC_DESCR;



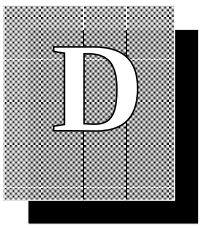
/* ------------------------------------------------------------------------ */

typedef struct T_FDL_STATISTIC_CTR
{
USIGN32  frame_send_count;
USIGN16         retry_count;
USIGN32         sd_count;
USIGN16         sd_error_count;
} T_FDL_STATISTIC_CTR;


#endif /*_PBL2TYPE_HEADER_*/

/* ---------------------- END OF FILE ---------------------------*/
```

 © 1995 PEP Modular Computers



D. DEMO EXAMPLES

```
/*    demo_M  <dst_station> <sap>
*     for example: demo_M 2 2
*!-------------------------------------------------------------------------!
*!                                                                         !
*!   Revision History:                                                     !
*!    #                     Reason                   By      Date        !
*!   ---  ---------------------------------------  ---   --------       !
*!     1  Original                                 HAH   11/20/92       !
*!        Library functions from pbL2hlf are used.                         !
*!        The function 'send_SRD ()' is used to send                       !
*!        and reply data from a remote station.                            !
*!        This application works in conjunction with                       !
*!        the program 'demo_S' which must run on the                       !
*!        remote station.                                                  !
*!        For the VM30 we allocate memory for the                          !
*!        output buffer in the TPRAM (colored memory),                     !
*!        so the value for the "D_MemAcc" parameter                        !
*!        could be set to zero in the device descriptor                    !
*!        for the PROFIBUS device /profi_<x>.                              !
*!                                                                         !
*!-------------------------------------------------------------------------!
*/


#include <stdio.h>
#include <errno.h>
#include <mem_pep.h>
#include <setsys.h>
#include <signal.h>
#include <time.h>

#include <pbL2con.h>
#include <pbL2type.h>
#include <pbL2hlf.h>

/* LOCAL DEFINES */

#define NUM_ARG   3                        /* Number of task arguments */

#define NOT_DONE  0
#define DONE      -1

#define TERMINATE           goto TERMLBL
#define SEND_BUF_LEN        255
#define ERROR     -1
```

 © 1995 PEP Modular Computers

```
/* FUNCTION_DECLARATIONS */

extern    USIGN32 open_PROFI ();
extern    USIGN32 close_PROFI ();
extern    USIGN32 open_JOB ();
extern    USIGN32 close_JOB ();
extern    USIGN32 send_SRD ();

void signal_handler ();

JOB_DESCR Job_Descr [1];

USIGN8    Dst_Station;
USIGN8    Dst_SAP;
USIGN8    Src_SAP;

USIGN8    Flag_Signal;
USIGN32   Signal;


/* FUNCTIONAL_DESCRIPTION */

/*-------------------------------------------------------------------------*/
/*  Function    main (argc, argv)                                          */
/*                                                                         */
/*-------------------------------------------------------------------------*/

#ifdef _UCC
main(int argc, char **argv )
#else
main( argc, argv ) int argc; char **argv;
#endif

{

JOB_DESCR *job_descr;

USIGN32 i;
USIGN32 input;
USIGN8 inchar;
USIGN8 * write_buf;
USIGN8 write_len;


USIGN8 * read_buf;
USIGN8 read_len;

USIGN8 * time_buf;
time_t time_tbl;

USIGN8  flag_open_profi, flag_open_job;
USIGN8    job_id;
USIGN32 status;
USIGN32 mem_type;
```

 © 1995 PEP Modular Computers

```
if (argc != NUM_ARG)
exit (E_PARAM);

/* ------------------------------------------------------------------------ */
/* - Get DESTINATION station and SAP                                      - */
/* ------------------------------------------------------------------------ */

/* get destination station */
sscanf(argv[1],"%d",&input);
Dst_Station = (USIGN8) (input);

/* get SAP for source and destination station */
sscanf(argv[2],"%d",&input);
Src_SAP = (USIGN8) (input);
Dst_SAP = Src_SAP;


Flag_Signal       = FALSE;


/*-------------------------------------------------------------------------*/
/* Install signal handler                                                  */
/*-------------------------------------------------------------------------*/

intercept(signal_handler);


/*-------------------------------------------------------------------------*/
/* Allocate memory for output buffer                                       */
/*-------------------------------------------------------------------------*/

if (_getsys(D_MPUType,sizeof(D_MPUType)) == 68030)
mem_type = TPRAM;
else
mem_type = 0;

if ((write_buf = (USIGN8 * ) srqcmem
     (SEND_BUF_LEN * sizeof (USIGN8), mem_type)) == (USIGN8 *) ERROR)
{
status = ERROR;
TERMINATE;
}


/*-------------------------------------------------------------------------*/
/* Open PROFIBUS device                                                    */
/* IF error                                                                */
/*    Goto TERMINATE                                                       */
/* ENDIF                                                                   */
/*-------------------------------------------------------------------------*/

if ((status = open_PROFI ()) == ERROR)
TERMINATE;
flag_open_profi = DONE;
```

 © 1995 PEP Modular Computers

```
/*-------------------------------------------------------------------------*/
/* prepare entries in Job Descriptor:                                      */
/*                                                                         */
/* job_id:                  job_descr.job                                  */
/* source SAP:              job_descr.ssap                                 */
/* destination SAP:         job_descr.dsap                                 */
/* number of IND buffer:    job_descr.nr_indbuf                            */
/*-------------------------------------------------------------------------*/

job_descr = &Job_Descr[0];

job_descr->job_id           = 0;
job_descr->ssap             = Src_SAP;
job_descr->nr_indbuf        = 0;

if ((status = open_JOB_S (job_descr)) != NULL)
TERMINATE;
flag_open_job = DONE;


/*-------------------------------------------------------------------------*/
/* MAIN loop begins here                                                   */
/*-------------------------------------------------------------------------*/

while (TRUE)
{
if (Flag_Signal)
{
     errno = Signal;
     status = ERROR;
     TERMINATE;
}

     sleep (1);

/*-------------------------------------------------------------------------*/
/* Prepare buffer for output                                               */
/*-------------------------------------------------------------------------*/

write_len = 26;
time (&time_tbl);
time_buf = (USIGN8 *) ctime(&time_tbl);

memcpy ((write_buf + FDL_OFFSET), time_buf, write_len);


/*-------------------------------------------------------------------------*/
/* prepare entries in Job Descriptor:                                      */
/*                                                                         */
/* remote station address:  job_descr.remote_station                       */
/* send buffer:             job_descr.send_buf                             */
/* send length:             job_descr.send_len                             */
/* send class:              job_descr.send_class                           */
/*-------------------------------------------------------------------------*/
```

 © 1995 PEP Modular Computers

```
job_id = 0;
job_descr = &Job_Descr[job_id];

job_descr->remote_station    = Dst_Station;
job_descr->dsap              = Dst_SAP;
job_descr->send_buf                        = write_buf;
job_descr->send_len                        = write_len;
job_descr->send_class                      = HIGH;

printf ("Send SRD to Job %d:    ",job_id);
status = send_SRD (job_id);

if (status == ERROR)         printf ("SYSTEM ERROR\n");
else
{
     switch ((USIGN8)status)
     {
          case NULL:         printf ("DONE\n");
                             break;
          case RR: printf ("RETRY\n");
                             break;

          case DH:
          case DL: printf ("SRD DONE, Receive DATA available\n");
                   printf ("                              ");
                   job_descr  = &Job_Descr[job_id];
                   read_buf   = job_descr->rec_buf;
                   read_len   = job_descr->rec_len;
                   for (i = 0; i < read_len; i++)
                   putchar(read_buf[i]);
                             break;

          case RDH:
          case RDL:          printf ("SRD ERROR, Receive DATA available\n");
                   printf ("                               ");
                   job_descr  = &Job_Descr[job_id];
                   read_buf   = job_descr->rec_buf;
                   read_len   = job_descr->rec_len;
                   for (i = 0; i < read_len; i++)
                   putchar(read_buf[i]);
                             break;

          case NR: printf ("SRD DONE, NO Receive DATA available\n");
                             break;

          default: printf ("ERROR status = %d\n",status);
     }
}
}                                        /* end: while (TRUE) */
```

 © 1995 PEP Modular Computers

```
TERMLBL:

if (flag_open_job == (USIGN8) DONE) close_JOB (job_id);
if (flag_open_profi == (USIGN8) DONE) close_PROFI();

if (status != ERROR)
errno = (status | 0x8000);

exit (errno);
}


/*---------------------------------------------------------------------------*/
/*   Functionbody  signal_handler (signal)                                   */
/*                                                                           */
/*---------------------------------------------------------------------------*/

#ifdef _UCC
void signal_handler (int signal)
#else
void signal_handler (signal)  int signal;
#endif

{
Flag_Signal = TRUE;
Signal = signal;
return;
}
```

 © 1995 PEP Modular Computers

```
/*    demo_S  <dst_station> <sap>
*     for example: demo_S 2 2
*!-------------------------------------------------------------------------!
*!                                                                         !
*!   Revision History:                                                     !
*!    #                     Reason                   By      Date         !
*!   ---  ---------------------------------------  ---   --------        !
*!     1  Original                                 HAH   20/DEC/92       !
*!        Library functions from pbL2hlf are used.                         !
*!        the function 'send_RPLUPD_S()' is used                           !
*!        to send data to a remote station.                                !
*!        This application works in conjunction with                       !
*!        the program 'demo_M' which must run on the                       !
*!        remote station.                                                  !
*!        For the VM30 we allocate memory for the                          !
*!        output buffer in the TPRAM (colored memory),                     !
*!        so the value for the "D_MemAcc" parameter                        !
*!        could be set to zero in the device descriptor                    !
*!        for the PROFIBUS device /profi_<x>.                              !
*!                                                                         !
*!     2  Toggle write buffer for function         HAH   09/JUN/93       !
*!        'send_RPLUPD_S ()'.                                               !
*!                                                                         !
*!-------------------------------------------------------------------------!
*/

#include <stdio.h>
#include <errno.h>
#include <mem_pep.h>
#include <setsys.h>
#include <signal.h>
#include <time.h>

#include <pbL2con.h>
#include <pbL2type.h>
#include <pbL2hlf.h>


/* LOCAL DEFINES */

#define NUM_ARG   3                       /* Number of task arguments */

#define NOT_DONE  0
#define DONE      -1

#define TERMINATE           goto TERMLBL
#define SEND_BUF_LEN        255
#define NR_OF_RESRC         0x2
#define ERROR               -1
```

```
/* FUNCTION_DECLARATIONS */

extern    USIGN32 open_PROFI ();
extern    USIGN32 close_PROFI ();
extern    USIGN32 open_JOB ();
extern    USIGN32 close_JOB ();
extern    USIGN32 send_RPLUPD_S ();
extern    USIGN32 receive_IND ();
extern    USIGN32 ready_IND ();
extern    USIGN32 release_IND ();

void signal_handler          (signal_code);


JOB_DESCR Job_Descr [1];

USIGN8  Dst_Station;
USIGN8    Dst_SAP;
USIGN8    Src_SAP;

USIGN8  Flag_Signal;
USIGN32   Signal;


/* FUNCTIONAL_DESCRIPTION */

/*-------------------------------------------------------------------------*/
/*  Function    main (argc, argv)                                          */
/*                                                                         */
/*-------------------------------------------------------------------------*/


#ifdef _UCC
main(int argc, char **argv )
#else
main( argc, argv ) int argc; char **argv;
#endif

{

JOB_DESCR *job_descr;

USIGN32 i;
USIGN32 input;
USIGN8 inchar;
USIGN8 * write_buf;
USIGN8 write_len;
BOOL   write_toggle;

USIGN8 * read_buf;
USIGN8 read_len;

USIGN8 * time_buf;
time_t time_tbl;
```

 © 1995 PEP Modular Computers

```
USIGN8  flag_open_profi, flag_open_job;
USIGN8    job_id;
USIGN32 status;
USIGN8    ind_service, ind_status;
USIGN32 mem_type;


if (argc != NUM_ARG)
exit (E_PARAM);

/* ------------------------------------------------------------------------ */
/* - Get DESTINATION station and SAP                                      - */
/* ------------------------------------------------------------------------ */

/* get destination station */
sscanf(argv[1],"%d",&input);
Dst_Station = (USIGN8) (input);

/* get SAP for source and destination station */
sscanf(argv[2],"%d",&input);
Src_SAP = (USIGN8) (input);
Dst_SAP = Src_SAP;


Flag_Signal       = FALSE;



/*-------------------------------------------------------------------------*/
/* Install signal handler                                                  */
/*-------------------------------------------------------------------------*/

intercept(signal_handler);


/*-------------------------------------------------------------------------*/
/* Allocate memory for output buffer                                       */
/*-------------------------------------------------------------------------*/

if (_getsys(D_MPUType,sizeof(D_MPUType)) == 68030)
mem_type = TPRAM;
else
mem_type = 0;

if ((write_buf = (USIGN8 * ) srqcmem
     (SEND_BUF_LEN * 2 * sizeof (USIGN8), mem_type)) == (USIGN8 *) ERROR)
{
status = ERROR;
TERMINATE;
}
```

 © 1995 PEP Modular Computers

```
/*------------------------------------------------------------------------*/
/* Open PROFIBUS device                                                   */
/* IF error                                                               */
/*    Goto TERMINATE                                                      */
/* ENDIF                                                                  */
/*------------------------------------------------------------------------*/

if ((status = open_PROFI ()) == ERROR)
TERMINATE;
flag_open_profi = DONE;


/*------------------------------------------------------------------------*/
/* prepare entries in Job Descriptor:                                     */
/*                                                                          */
/* job_id:                  job_descr.job                                 */
/* source SAP:              job_descr.ssap                                */
/* destination SAP:         job_descr.dsap                                */
/* number of IND buffer:    job_descr.nr_indbuf                           */
/*------------------------------------------------------------------------*/

job_descr = &Job_Descr[0];

job_descr->job_id           = 0;
job_descr->ssap             = Src_SAP;
job_descr->nr_indbuf        = NR_OF_RESRC;

if ((status = open_JOB_R_SRD (job_descr)) != NULL)
TERMINATE;
flag_open_job = DONE;

write_toggle      = 0;

/*------------------------------------------------------------------------*/
/* MAIN loop begins here                                                  */
/*------------------------------------------------------------------------*/

while (TRUE)
{

if (Flag_Signal)
{
     errno = Signal;
     status = ERROR;
     TERMINATE;
}
```

 © 1995 PEP Modular Computers

```
/*-------------------------------------------------------------------------*/
/* Prepare buffer for output                                               */
/*-------------------------------------------------------------------------*/

time (&time_tbl);
time_buf = (USIGN8 *) ctime(&time_tbl);
write_len = 26;

memcpy ((write_buf + write_toggle * SEND_BUF_LEN + FDL_OFFSET),
          time_buf, write_len);


/*-------------------------------------------------------------------------*/
/* prepare entries in Job Descriptor:                                      */
/*                                                                           */
/* remote station address:   job_descr.remote_station                      */
/* send buffer:              job_descr.send_buf                            */
/* send length:              job_descr.send_len                            */
/* send class:               job_descr.send_class                          */
/*-------------------------------------------------------------------------*/

job_id = 0;
job_descr = &Job_Descr[job_id];

job_descr->remote_station    = Dst_Station;
job_descr->dsap              = Dst_SAP;
job_descr->send_buf               = write_buf + write_toggle * SEND_BUF_LEN;
job_descr->send_len               = write_len;
job_descr->send_class             = HIGH;

printf ("Send REPLY_UPDATE to Job %d:    \n",job_id);

do
{
     status = send_RPLUPD_S (job_id);
     if (status == ERROR) TERMINATE;
}    while (status);

write_toggle      = (write_toggle + 1) & 0x01;

/*-------------------------------------------------------------------------*/
/* wait for SRD Indication                                                 */
/*-------------------------------------------------------------------------*/

status = receive_IND();
if (status == ERROR) TERMINATE;

else
{
     job_id                  = (USIGN8) status;
     job_descr    = &Job_Descr[job_id];
     read_buf     = job_descr->ind_buf;
     read_len     = job_descr->ind_len;
     ind_status   = job_descr->status;
     ind_service  = job_descr->service;
```

 © 1995 PEP Modular Computers

```
     printf ("Received SRD from Job %d,  Update status = %d\n",
                 job_id,ind_status);
     if ((ind_status == LO) || (ind_status == HI))
     {
          printf ("SRD Data:  ",ind_status);
          for (i = 0; i < read_len; i++)
                  putchar(read_buf[i]);
     }

     release_IND (job_id);
}

}                                          /* end: while (TRUE) */



TERMLBL:

if (flag_open_job == (USIGN8) DONE) close_JOB (job_id);
if (flag_open_profi == (USIGN8) DONE) close_PROFI();

if (status != ERROR)
errno = (status | 0x8000);

exit (errno);
}

/*---------------------------------------------------------------------------*/
/*   Functionbody  signal_handler (signal)                                   */
/*                                                                           */
/*---------------------------------------------------------------------------*/


#ifdef _UCC
void signal_handler (int signal)
#else
void signal_handler (signal)  int signal;
#endif

{
Flag_Signal = TRUE;
Signal = signal;
return;
}
```

 © 1995 PEP Modular Computers

```
/*    sda_demo <PROFIBUS device_name> <own_station> <dst_station> <sap>
     for example: sda_demo /profi_1 1 2 3
!-------------------------------------------------------------------------!
!                                                                         !
!   Revision History:                                                     !
!    #                     Reason                      By      Date     !
!   ---  ------------------------------------------  ---   --------  !
!     1  Original                                    HAH   10/15/92  !
!        NET-descriptor is used as a data module to                       !
!        extract information about PROFIBUS device                        !
!        and bus parameter                                                !
!     2  Changed due to new structure of NET-        HAH   12/03/92  !
!        descriptor, NET-descriptor no longer                             !
!        used. Bus parameters defined directly                            !
!        in function "set_busparameter".                                  !
!        For the VM30 we allocate memory for the                          !
!        output buffer in the TPRAM (colored memory),                     !
!        so the value for the "D_MemAcc" parameter                        !
!        could be set to zero in the device descriptor                    !
!        for the PROFIBUS device /profi_<x>.                              !
!                                                                         !
!-------------------------------------------------------------------------!
*/

@_sysedit:        equ       2

#include <stdio.h>
#include <errno.h>
#include <mem_pep.h>
#include <setsys.h>
#include <signal.h>
#include <time.h>

#include <pbL2con.h>
#include <pbL2type.h>




/* LOCAL DEFINES */

#define NUM_ARG   0x05                    /* Number of task arguments */

#define NOT_DONE  0
#define DONE      -1

#define NO_WAIT_CON 0x00
#define WAIT_CON    0xff

#define TERMINATE           goto TERMLBL
#define   USR_BUF_LEN       242
#define IND_BUF_LEN         255
#define SEND_BUF_LEN        255
#define NR_OF_RESRC         0x2
#define ERROR               -1
```

```
#define STDIN               0
#define STDOUT              1


/* FUNCTION_DECLARATIONS */

extern T_FDL_SERVICE_DESCR * fdl_con_ind ();
extern T_FDL_SERVICE_DESCR * fdl_con_ind_poll ();
extern USIGN32 fdl_req ();
extern USIGN32 fdl_open ();
extern USIGN32 fdl_close ();

void signal_handler         (signal_code);

USIGN32 set_busparameter ();
USIGN32 activate_sap ();
USIGN32 deactivate_sap ();
USIGN32 put_resrc_to_sap ();
USIGN32 withdraw_resrc_from_sap ();
USIGN32 do_sda_ind (sdb_ptr);
USIGN32 alloc_service_mem ();
USIGN32 alloc_mem_for_service_descr ();
USIGN32 alloc_mem_for_buspar ();
USIGN32 alloc_mem_for_sap ();
USIGN32 alloc_mem_for_receive_data ();
USIGN32 alloc_mem_for_sda_req ();
VOID *memory_allocation (length);
VOID memory_deallocation (ptr);
VOID block_copy (source, desc, length);

USIGN32 WriteOutput (wr_buf, wr_len);
USIGN32 ReadInput (rd_buf, rd_len);


/* LOCAL_DATA */


char *dev_name;

/* init control flags                                          */
BOOL flag_fdl_open;                     /* open PROFIBUS device done */
BOOL flag_put_resrc;                    /* service PUT_RESRC_TO_FDL done*/
BOOL flag_activate_sap;                 /* service FMA2_ACTIVATE_SAP done*/

/* flow control flags                                          */
BOOL flag_signal;                       /* signal received           */
BOOL flag_wait_resrccon;
BOOL flag_wait_sdacon;
int flag;
```

 © 1995 PEP Modular Computers

```
T_FDL_SERVICE_DESCR    * sdb_ptr;
T_FDL_SERVICE_DESCR    * usr_sdb_ptr;
T_FDL_SERVICE_DESCR    * sda_sdb_ptr;
T_FDL_SERVICE_DESCR    * resrc_sdb_ptr;
T_FDL_SERVICE_DESCR    * rec_resrc_ptr;
T_FDL_SERVICE_DESCR    * resrc_parklist[NR_OF_RESRC];
T_FDL_SERVICE_DESCR    * withdr_resrc_sdb_ptr;
T_FDL_RESRC_DESCR      * withdr_resrc_descr_ptr;
T_FDL_RESRC_DESCR      * resrc_descr_ptr;
T_FDL_SAP_DESCR        * sap_descr_ptr;
T_FDL_SAP_BLOCK        * sap_block_ptr;
T_FDL_SR_BLOCK         * send_sr_block;
T_FDL_SR_BLOCK         * rec_sr_block;
T_BUSPAR_BLOCK         * buspar_ptr;


struct    RD_BUF_BLOCK
{
     USIGN8 len;
     T_FDL_SERVICE_DESCR *buffer_ptr;
     struct RD_BUF_BLOCK * next_ptr;
};
typedef struct RD_BUF_BLOCK RD_BUF_BLOCK;
RD_BUF_BLOCK      *rd_backup_ptr, *free_backup_ptr, *full_backup_ptr;

USIGN8 buffer [128];

USIGN8 * read_buf;
USIGN8 read_len;

USIGN8 * send_buf;
USIGN8 * write_buf;
USIGN8 write_len;

USIGN8 * time_buf;
time_t time_tbl;

USIGN8 resrc_cnt;


int signal;
int input;
char inchar;

USIGN8  own_station;
USIGN8  dst_station;

USIGN8    own_sap;
USIGN8    dst_sap;
```

```
/* FUNCTIONAL_DESCRIPTION */

/*-------------------------------------------------------------------------*/
/*  Function    main (argc, argv)                                          */
/*                                                                         */
/*-------------------------------------------------------------------------*/

main( argc, argv )
int argc;
char **argv;

{

int i;


if (argc != NUM_ARG)
exit (E_PARAM);

/* ----------------------------------------------------------------------- */
/* - Define SOURCE and DESTINATION station                               - */
/* ----------------------------------------------------------------------- */

 /* get source station */
 sscanf(argv[2],"%d",&input);
 own_station = (USIGN8) (input);
 /* get destination station */
 sscanf(argv[3],"%d",&input);
 dst_station = (USIGN8) (input);

 /* get SAP for source and destination station */
 sscanf(argv[4],"%d",&input);
 own_sap = (USIGN8) (input);
 dst_sap = own_sap;

dev_name = argv[1];


/*-------------------------------------------------------------------------*/
/* Predefine several flags                                                 */
/*-------------------------------------------------------------------------*/

flag_fdl_open               = NOT_DONE;
flag_put_resrc              = NOT_DONE;
flag_activate_sap           = NOT_DONE;

flag_signal                 = FALSE;
flag_wait_resrccon          = FALSE;
flag_wait_sdacon            = FALSE;
```

 © 1995 PEP Modular Computers

```
/*-------------------------------------------------------------------------*/
/* Install signal handler                                                  */
/*-------------------------------------------------------------------------*/

intercept(signal_handler);


/*-------------------------------------------------------------------------*/
/* Open PROFIBUS device                                                    */
/* IF error                                                                */
/*    Goto TERMINATE                                                       */
/* ENDIF                                                                   */
/*-------------------------------------------------------------------------*/

if (fdl_open (dev_name) == -1)
TERMINATE;

flag_fdl_open = DONE;


if (alloc_service_mem () == ERROR)
TERMINATE;


if (set_busparameter () == ERROR)
TERMINATE;

if (activate_sap () == ERROR)
TERMINATE;

flag_activate_sap = DONE;

resrc_cnt = NR_OF_RESRC;
if (put_resrc_to_sap (WAIT_CON) == ERROR)
TERMINATE;

flag_put_resrc = DONE;

write_buf = &buffer[0];

/*-------------------------------------------------------------------------*/
/* MAIN loop begins here                                                   */
/*-------------------------------------------------------------------------*/

while (TRUE)
{
sleep (1);

time (&time_tbl);
time_buf = (USIGN8 *) ctime(&time_tbl);
write_len = 26;
block_copy (time_buf, write_buf, write_len);
```

 © 1995 PEP Modular Computers

```
if (flag_signal)
{
     if (signal == SIGINT)
     {
          printf ("\nINPUT NEW LINE:  ");
          write_len = 0;
          inchar = 0;
          while (inchar != EOL)
          {
                  inchar = getchar ();
                  buffer[write_len++] = inchar;
          }
          flag_signal = FALSE;
     }
     else
          TERMINATE;
}

flag = ReadInput (read_buf, &read_len);
if (flag > NULL)
{
     printf ("Read Input:    ");
     for (i = 0; i < read_len; i++)
          putchar(read_buf[i]);
}
if (flag == ERROR) printf  ("Read Input:      ERROR\n");

flag = WriteOutput (write_buf,write_len);
if (flag == ERROR) printf ("Write Output:    ERROR\n");
if (flag == NULL)  printf ("Write Output:    RETRY\n");
if (flag > NULL)   printf ("Write Output:    DONE\n");


}                                          /* end: while (TRUE) */



TERMLBL:

/* if (flag_put_resrc == DONE) withdraw_resrc_from_sap (); */
if (flag_activate_sap == DONE) deactivate_sap ();
if (flag_fdl_open == DONE) fdl_close();

return (errno);
}



void signal_handler (signal_code)
      int signal_code;
```

 © 1995 PEP Modular Computers

```
/*-------------------------------------------------------------------------*/
/*   Functionbody  signal_handler (signal)                                 */
/*                                                                         */
/*-------------------------------------------------------------------------*/
{
flag_signal = TRUE;
signal = signal_code;
return;
}

/*************************************************************************
*                                                                        *
* FMA2/FDL Service Functions:                                            *
*                                                                        *
* set_busparameter ()                                                    *
* activate_sap ()                                                        *
* deactivate_sap ()                                                      *
* do_sda_ind ()                                                          *
* put_resrc_to_sap ()                                                    *
* withdraw_resrc_from_sap ()                                             *
*                                                                        *
* WriteOutput ()                                                         *
* ReadInput ()                                                           *
*                                                                        *
*************************************************************************/


USIGN32 set_busparameter ()

/*-------------------------------------------------------------------------
FUNCTIONAL_DESCRIPTION
This function fills the busparameter block with desired values and puts it
to layer2.
-------------------------------------------------------------------------*/
{

USIGN32 ret_val = NULL;


/* The following parameters are dependent on the selected baud rate */
/* ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ */

buspar_ptr->loc_add.station  = own_station;
buspar_ptr->loc_add.segment  = NO_SEGMENT;
buspar_ptr->baud_rate        = K_BAUD_187_5;
```

 © 1995 PEP Modular Computers

```
switch ( (USIGN8) buspar_ptr->baud_rate )
{
case K_BAUD_500 :

          buspar_ptr->tsl    = 4000;
          buspar_ptr->min_tsdr             = 100;
          buspar_ptr->max_tsdr             = 2000;
          buspar_ptr->tqui   = 0;
          buspar_ptr->tset   = 50;
          buspar_ptr->ttr    = 50000;
          buspar_ptr->g      = 2;
                  break ;

case K_BAUD_187_5 :

          buspar_ptr->tsl    = 2000;
          buspar_ptr->min_tsdr             = 40;
          buspar_ptr->max_tsdr             = 1000;
          buspar_ptr->tqui   = 0;
          buspar_ptr->tset   = 20;
          buspar_ptr->ttr    = 25000;
          buspar_ptr->g      = 2;
                  break ;

case K_BAUD_93_75 :

          buspar_ptr->tsl    = 1000;
          buspar_ptr->min_tsdr             = 25;
          buspar_ptr->max_tsdr             = 500;
          buspar_ptr->tqui   = 0;
          buspar_ptr->tset   = 40;
          buspar_ptr->ttr    = 13000;
          buspar_ptr->g      = 2;
                  break ;

case K_BAUD_19_2 :

          buspar_ptr->tsl    = 400;
          buspar_ptr->min_tsdr             = 10;
          buspar_ptr->max_tsdr             = 200;
          buspar_ptr->tqui   = 0;
          buspar_ptr->tset   = 4;
          buspar_ptr->ttr    = 9000;
          buspar_ptr->g      = 2;
                  break ;

case K_BAUD_9_6 :
default         :
          buspar_ptr->tsl    = 400;
          buspar_ptr->min_tsdr             = 20;
          buspar_ptr->max_tsdr             = 100;
          buspar_ptr->tqui   = 0;
          buspar_ptr->tset   = 2;
          buspar_ptr->ttr    = 5000;
          buspar_ptr->g      = 2;
                  break ;
```

 © 1995 PEP Modular Computers

```
}

buspar_ptr->hsa              = 10;
buspar_ptr->medium_red       = NO_REDUNDANCY;
buspar_ptr->in_ring_desired  = TRUE;
buspar_ptr->max_retry_limit  = 2;

buspar_ptr->ident[0]         = 13;
buspar_ptr->ident[1]         = 1;
buspar_ptr->ident[2]         = 1;
buspar_ptr->ident[3]         = 1;
buspar_ptr->ident[4]         = 'P';
buspar_ptr->ident[5]         = 'R';
buspar_ptr->ident[6]         = 'O';
buspar_ptr->ident[7]         = 'F';
buspar_ptr->ident[8]         = 'I';
buspar_ptr->ident[9]         = 'B';
buspar_ptr->ident[10]        = 'U';
buspar_ptr->ident[11]        = 'S';
buspar_ptr->ident[12]        = ' ';
buspar_ptr->ident[13]        = 'U';
buspar_ptr->ident[14]        = 'S';
buspar_ptr->ident[15]        = 'E';
buspar_ptr->ident[16]        = 'R';
buspar_ptr->ident[17]        = '1';
buspar_ptr->ident[18]        = '1';
buspar_ptr->ident[19]        = '1';
buspar_ptr->ind_buf_len      = 0;

usr_sdb_ptr->sap             = MSAP_0;
usr_sdb_ptr->service         = FMA2_SET_BUSPARAMETER;
usr_sdb_ptr->primitive       = REQ;
usr_sdb_ptr->descr_ptr       = (USIGN8 *)buspar_ptr;

if (fdl_req(usr_sdb_ptr) == ERROR)


sdb_ptr = fdl_con_ind ();
if ( ((USIGN32) sdb_ptr == ERROR) || ((USIGN32) sdb_ptr == NULL) )
return (ERROR);

if ((sdb_ptr->status != OK) && (sdb_ptr != LR))
{
errno = E_PARAM;
return (ERROR);
}

return (ret_val);
}
```

```
USIGN32 activate_sap ()
/*---------------------------------------------------------------------------
FUNCTIONAL_DESCRIPTION
This function activates a sap for desired action. To activate a sap for
responding a SRD or CSRD request you need the function activate_rsap().
---------------------------------------------------------------------------*/

{

USIGN32 ret_val = 0;


(T_FDL_SAP_DESCR *) usr_sdb_ptr->descr_ptr      = sap_descr_ptr;


usr_sdb_ptr->sap                            = MSAP_2;
usr_sdb_ptr->service                        = FMA2_ACTIVATE_SAP;
usr_sdb_ptr->primitive                      = REQ;

sap_descr_ptr->sap_nr                       = own_sap;
sap_descr_ptr->rem_add.station              = dst_station;
sap_descr_ptr->rem_add.segment              = NO_SEGMENT;

sap_block_ptr->max_len_sda_req_low = IND_BUF_LEN - (FDL_OFFSET + FDL_TRAILER);
sap_block_ptr->max_len_sda_req_high = IND_BUF_LEN - (FDL_OFFSET + FDL_TRAILER);
sap_block_ptr->max_len_sdn_req_low = IND_BUF_LEN - (FDL_OFFSET + FDL_TRAILER);
sap_block_ptr->max_len_sdn_req_high = IND_BUF_LEN - (FDL_OFFSET + FDL_TRAILER);
sap_block_ptr->max_len_srd_req_low = IND_BUF_LEN - (FDL_OFFSET + FDL_TRAILER);
sap_block_ptr->max_len_srd_req_high = IND_BUF_LEN - (FDL_OFFSET + FDL_TRAILER);
sap_block_ptr->max_len_sda_ind_low = IND_BUF_LEN - (FDL_OFFSET + FDL_TRAILER);
sap_block_ptr->max_len_sda_ind_high = IND_BUF_LEN - (FDL_OFFSET + FDL_TRAILER);
sap_block_ptr->max_len_sdn_ind_low = IND_BUF_LEN - (FDL_OFFSET + FDL_TRAILER);
sap_block_ptr->max_len_sdn_ind_high = IND_BUF_LEN - (FDL_OFFSET + FDL_TRAILER);
sap_block_ptr->max_len_srd_con_low = IND_BUF_LEN - (FDL_OFFSET + FDL_TRAILER);
sap_block_ptr->max_len_srd_con_high = IND_BUF_LEN - (FDL_OFFSET + FDL_TRAILER);

sap_descr_ptr->sda = BOTH_ROLES;
sap_descr_ptr->sdn = BOTH_ROLES;
sap_descr_ptr->srd = SERVICE_NOT_ACTIVATED;
sap_descr_ptr->csrd = SERVICE_NOT_ACTIVATED;

if (ret_val = fdl_req(usr_sdb_ptr) == ERROR)
return (ERROR);

sdb_ptr = fdl_con_ind ();
if ( ((USIGN32) sdb_ptr == ERROR) || ((USIGN32) sdb_ptr == NULL) )
return (ERROR);

if (sdb_ptr->status != OK)
{
errno = E_PARAM;
return (ERROR);
}

return(ret_val);
}
```

 © 1995 PEP Modular Computers

```
USIGN32 deactivate_sap ()

/*-----------------------------------------------------------------------------
FUNCTIONAL_DESCRIPTION
This function deactivates a sap for desired action.
-----------------------------------------------------------------------------*/
{

T_FDL_DEACT_SAP sap_deact;
T_FDL_DEACT_SAP *sap_deact_ptr;
USIGN32 ret_val = 0;


sap_deact_ptr =  &sap_deact;
(T_FDL_DEACT_SAP *) usr_sdb_ptr->descr_ptr      = sap_deact_ptr;


usr_sdb_ptr->sap                            = MSAP_2;
usr_sdb_ptr->service                        = FMA2_DEACTIVATE_SAP;
usr_sdb_ptr->primitive                      = REQ;

sap_deact_ptr->ssap                         = own_sap;


if (ret_val = fdl_req(usr_sdb_ptr) == ERROR)
return (ERROR);

while (TRUE)
{
sdb_ptr = fdl_con_ind ();
if ( ((USIGN32) sdb_ptr == ERROR) || ((USIGN32) sdb_ptr == NULL) )
     return (ERROR);
if (sdb_ptr->service == FMA2_DEACTIVATE_SAP)
     return(ret_val);
}

}
```

 © 1995 PEP Modular Computers

```
USIGN32 WriteOutput (wr_buf,wr_len)
USIGN8 *wr_buf;
USIGN8 wr_len;

/*---------------------------------------------------------------------------
FUNCTIONAL_DESCRIPTION
This function creates an SDA request by filling
    the T_FDL_SERVICE_DESCR block
    the T_FDL_SR_BLOCK
    the telegram buffer with desired message.

---------------------------------------------------------------------------*/


{

USIGN32   ret_val = NULL;


block_copy (wr_buf, &send_buf[FDL_OFFSET], wr_len);

sda_sdb_ptr->service                  = SDA;
sda_sdb_ptr->primitive                = REQ;
sda_sdb_ptr->sap                      = own_sap;
sda_sdb_ptr->descr_ptr = (USIGN8 *)send_sr_block;

send_sr_block->rem_add.station        = dst_station;
send_sr_block->remote_sap             = dst_sap;
send_sr_block->rem_add.segment        = NO_SEGMENT;
send_sr_block->serv_class             = HIGH;
send_sr_block->send_data.length       = wr_len;
send_sr_block->send_data.buffer_ptr   = send_buf;

if ((ret_val = fdl_req(sda_sdb_ptr)) == ERROR)
return (ret_val);

flag_wait_sdacon = TRUE;

while (flag_wait_sdacon)
{
sdb_ptr = fdl_con_ind ();

if ( ((USIGN32) sdb_ptr != NULL) && ((USIGN32) sdb_ptr != -1) )
{
     if ((sdb_ptr->primitive == CON) && (sdb_ptr->service == SDA))
     {
          flag_wait_sdacon = FALSE;
          if (sdb_ptr->status == OK)       return (wr_len);

          else              /* sdb_ptr->status != OK */
          {
                  if (sdb_ptr->status == RR)                return (NULL);
                  else                                      return (ERROR);
          }
     }
```

 © 1995 PEP Modular Computers

```
     else
          do_sda_ind (sdb_ptr);
}

else return (ERROR);

}                 /* end: while (flag_wait_sdacon) */

}

/*-------------------------------------------------------------------------
USIGN32 ReadInput ()

FUNCTIONAL_DESCRIPTION
This function handles an SDA or SDN indication by
    filling the provided buffer with the incoming message

-------------------------------------------------------------------------*/

USIGN32 ReadInput (rd_buf, rd_len)
USIGN8 *rd_buf;
USIGN8 *rd_len;

{

T_FDL_SERVICE_DESCR * sdb_ptr;
USIGN8 * buf;
USIGN8 len;
USIGN32   ret_val = NULL;

if (full_backup_ptr->len == NULL)
{
sdb_ptr = fdl_con_ind_poll ();

if ( ((USIGN32) sdb_ptr != NULL) && ((USIGN32) sdb_ptr != -1) )
     do_sda_ind (sdb_ptr);
else
     return (NULL);
}

if (full_backup_ptr->len != NULL)
{
sdb_ptr = full_backup_ptr->buffer_ptr;
rec_sr_block = (T_FDL_SR_BLOCK *) sdb_ptr->descr_ptr;
buf  = rec_sr_block->receive_data.buffer_ptr;
len  = rec_sr_block->receive_data.length;

block_copy (buf, rd_buf, len);
full_backup_ptr->len = NULL;
full_backup_ptr = full_backup_ptr->next_ptr;

resrc_parklist[resrc_cnt++] = sdb_ptr;

if (put_resrc_to_sap (NO_WAIT_CON) == ERROR)
     return (ERROR);
```

```
flag_wait_resrccon = TRUE;
while (flag_wait_resrccon)
{

     sdb_ptr = fdl_con_ind ();

     if ( ((USIGN32) sdb_ptr != NULL) && ((USIGN32) sdb_ptr != -1) )
     {
          if (sdb_ptr->primitive == CON)
                              /* service = PUT_RESRC_TO_FDL */
          {
                   if (sdb_ptr->status == OK)
                              flag_wait_resrccon = FALSE;
          }                   /* end: service = PUT_RESRC_TO_FDL */


          else                /* primitive = IND */
                   do_sda_ind (sdb_ptr);
     }                        /* end: if (sdb_ptr != NULL/-1) *
     else
          return(ERROR);
}                             /* end: while (flag_wait_resrc_con) */

rd_len[0] = len;
return (len);

}                             /* end:  (full_backup_ptr->len != NULL) */


}



/*-----------------------------------------------------------------------------
USIGN32 do_sda_ind ()

FUNCTIONAL_DESCRIPTION
This function handles an SDA indication

-----------------------------------------------------------------------------*/

USIGN32 do_sda_ind (sdb_ind)
T_FDL_SERVICE_DESCR * sdb_ind;

{

USIGN32   ret_val = NULL;

rec_sr_block = (T_FDL_SR_BLOCK *) sdb_ind->descr_ptr;

free_backup_ptr->len =  rec_sr_block->receive_data.length;
free_backup_ptr->buffer_ptr = sdb_ind;
free_backup_ptr = free_backup_ptr->next_ptr;

return (ret_val);
}
```

 © 1995 PEP Modular Computers

```
/*-----------------------------------------------------------------------------

USIGN32 put_resrc_to_sap (wait_con)

FUNCTIONAL_DESCRIPTION
This function puts receive resources out of a list of allocated memory to the
activated sap
-----------------------------------------------------------------------------*/

USIGN32 put_resrc_to_sap (wait_con)

USIGN8 wait_con;


{

T_FDL_SERVICE_DESCR * current_resrc_ptr;
USIGN8 i;
USIGN32   ret_val = NULL;


usr_sdb_ptr->sap            = own_sap;
usr_sdb_ptr->service        = PUT_RESRC_TO_FDL;
usr_sdb_ptr->primitive      = REQ;


(T_FDL_RESRC_DESCR *)usr_sdb_ptr->descr_ptr= resrc_descr_ptr;

resrc_descr_ptr->nr_of_resources = resrc_cnt;

resrc_descr_ptr->dsap                   = dst_sap;
resrc_descr_ptr->rem_add.station        = dst_station;
resrc_descr_ptr->rem_add.segment        = NO_SEGMENT;

resrc_descr_ptr->resrc_ptr = resrc_parklist[0];
current_resrc_ptr = resrc_parklist[0];

i = 1;
while (i < resrc_cnt)
{
current_resrc_ptr->next_descr = resrc_parklist[i++];
current_resrc_ptr = current_resrc_ptr->next_descr;
}

current_resrc_ptr->next_descr = NULL;
resrc_cnt = 0;

if ((ret_val = fdl_req(usr_sdb_ptr)) == ERROR)
return (ERROR);

if (wait_con == WAIT_CON)
{
sdb_ptr = fdl_con_ind ();
if ( ((USIGN32) sdb_ptr == ERROR) || ((USIGN32) sdb_ptr == NULL) )
     return (ERROR);
```

 © 1995 PEP Modular Computers

```
if (sdb_ptr->status != OK)
{
     errno = E_PARAM;
     return (ERROR);
}

}

return (ret_val);
}



/*-------------------------------------------------------------------------
FUNCTIONAL_DESCRIPTION
Withdraw resources from SAP
-------------------------------------------------------------------------*/


USIGN32 withdraw_resrc_from_sap ()

{

USIGN32 ret_val = NULL;



withdr_resrc_sdb_ptr->sap                  = own_sap;
withdr_resrc_sdb_ptr->service              = WITHDRAW_RESRC_FROM_FDL;
withdr_resrc_sdb_ptr->primitive            = REQ;

(T_FDL_RESRC_DESCR *)withdr_resrc_sdb_ptr->descr_ptr
                                                                        =
withdr_resrc_descr_ptr;

withdr_resrc_descr_ptr->dsap                                            =
dst_sap;
withdr_resrc_descr_ptr->rem_add.station    = dst_station;
withdr_resrc_descr_ptr->rem_add.segment    = NO_SEGMENT;

if ((ret_val = fdl_req(withdr_resrc_sdb_ptr)) == ERROR)
return (ERROR);

sdb_ptr = fdl_con_ind ();
if ( ((USIGN32) sdb_ptr == ERROR) || ((USIGN32) sdb_ptr == NULL) )
return (ERROR);

return (ret_val);
}
```

 © 1995 PEP Modular Computers

```
/*****************************************************************************
*                                                                            *
* Memory Management Functions:                                               *
*                                                                            *
* alloc_service_mem ()                                                       *
* alloc_mem_for_service_descr ()                                             *
* alloc_mem_for_buspar ()                                                    *
* alloc_mem_for_sap ()                                                       *
* alloc_mem_for_receive_data ()                                              *
* alloc_mem_for_sda_req()                                                    *
*                                                                            *
*****************************************************************************/

USIGN32 alloc_service_mem ()

/*---------------------------------------------------------------------------
FUNCTIONAL_DESCRIPTION
This function calls the memory allocation functions.
---------------------------------------------------------------------------*/
{

if (alloc_mem_for_service_descr () == -1)  return (errno);
if (alloc_mem_for_buspar () == -1)         return (errno);
if (alloc_mem_for_sap () == -1)            return (errno);
if (alloc_mem_for_receive_data () == -1)   return (errno);
if (alloc_mem_for_sda_req() == -1)         return (errno);

return(NULL);
}



USIGN32 alloc_mem_for_service_descr ()
{

if ((usr_sdb_ptr = (T_FDL_SERVICE_DESCR *)
memory_allocation (sizeof (T_FDL_SERVICE_DESCR))) == NULL)
return (ERROR);
return (NULL);
}



USIGN32 alloc_mem_for_buspar ()
{

if ((buspar_ptr = (T_BUSPAR_BLOCK *)
     memory_allocation (sizeof (T_BUSPAR_BLOCK))) == NULL)
return (ERROR);

if ((buspar_ptr->ident = (USIGN8 *)
     memory_allocation(sizeof(USIGN8)*14)) == NULL)
return (ERROR);

return (NULL);
}
```

 © 1995 PEP Modular Computers

```
USIGN32 alloc_mem_for_sap ()
{

if ( (sap_descr_ptr = (T_FDL_SAP_DESCR *)
     memory_allocation (sizeof (T_FDL_SAP_DESCR))) == NULL )
return (ERROR);

if ((sap_block_ptr = (T_FDL_SAP_BLOCK *)
     memory_allocation (sizeof (T_FDL_SAP_BLOCK))) == NULL)
return (ERROR);

sap_descr_ptr->sap_block_ptr = (USIGN8 *) sap_block_ptr;
return (NULL);
}



USIGN32 alloc_mem_for_receive_data ()
{

T_FDL_SR_BLOCK    * sr_ptr;
USIGN8            * buf_ptr;
USIGN16   i;


if ((resrc_sdb_ptr = (T_FDL_SERVICE_DESCR *)
     memory_allocation (sizeof (T_FDL_SERVICE_DESCR))) == NULL)
return (ERROR);

if ((resrc_descr_ptr = (T_FDL_RESRC_DESCR *) memory_allocation
     (sizeof (T_FDL_RESRC_DESCR))) == NULL)
return (ERROR);

if ((withdr_resrc_sdb_ptr = (T_FDL_SERVICE_DESCR *)
     memory_allocation (sizeof (T_FDL_SERVICE_DESCR))) == NULL)
return (ERROR);

if ((withdr_resrc_descr_ptr = (T_FDL_RESRC_DESCR *) memory_allocation
     (sizeof (T_FDL_RESRC_DESCR))) == NULL)
return (ERROR);

if ((buf_ptr = memory_allocation (NR_OF_RESRC * IND_BUF_LEN)) == NULL)
return (ERROR);

if ((read_buf = memory_allocation (IND_BUF_LEN)) == NULL)
return (ERROR);

if ((sr_ptr = (T_FDL_SR_BLOCK *) memory_allocation
     (NR_OF_RESRC * sizeof (T_FDL_SR_BLOCK))) == NULL)
return (ERROR);

if ((rec_resrc_ptr = (T_FDL_SERVICE_DESCR *) memory_allocation
     (NR_OF_RESRC * sizeof (T_FDL_SERVICE_DESCR))) == NULL)
return (ERROR);
```

 © 1995 PEP Modular Computers

```
if ((rd_backup_ptr = (RD_BUF_BLOCK *) memory_allocation
     (NR_OF_RESRC * sizeof (RD_BUF_BLOCK))) == NULL)
return (ERROR);


for(i = 0; i < NR_OF_RESRC; i++)
{
sr_ptr[i].resource.buffer_ptr              = &buf_ptr[i * IND_BUF_LEN];
sr_ptr[i].resource.length    = IND_BUF_LEN;
(T_FDL_SR_BLOCK *) rec_resrc_ptr[i].descr_ptr = &sr_ptr[i];
resrc_parklist[i] = &rec_resrc_ptr[i];
rd_backup_ptr[i].len = NULL;
if ( i == (NR_OF_RESRC - 1) )
     rd_backup_ptr[i].next_ptr = &rd_backup_ptr[0];
else
     rd_backup_ptr[i].next_ptr = &rd_backup_ptr[i+1];
}

free_backup_ptr = rd_backup_ptr;
full_backup_ptr = rd_backup_ptr;

return (NULL);
}



USIGN32 alloc_mem_for_sda_req ()
{
USIGN32   mem_type;

if ((sda_sdb_ptr = (T_FDL_SERVICE_DESCR *)
     memory_allocation (sizeof (T_FDL_SERVICE_DESCR))) == NULL)
return (ERROR);

if ((send_sr_block =  (T_FDL_SR_BLOCK *)
     memory_allocation (sizeof (T_FDL_SR_BLOCK))) == NULL)
return (ERROR);

if (_getsys(D_MPUType,sizeof(D_MPUType)) == 68030)
mem_type = TPRAM;
else
mem_type = 0;

if ((send_buf = (USIGN8 * ) srqcmem
     (SEND_BUF_LEN * sizeof (USIGN8), mem_type)) == (USIGN8 *) ERROR)
return (ERROR);

send_sr_block->send_data.buffer_ptr        = send_buf;

return (NULL);
}
```

 © 1995 PEP Modular Computers

```
VOID *memory_allocation (length)
USIGN16 length;

/*****************************************************************/
/*  Functionbody  memory_allocation (length)                     */
/*                                                               */
/*  -->  With the parameter "length" memory space willbe demanded*/
/*                                                               */
/*  -->  The return value of the function is a pointer to the first*/
/*    Byte of the allocated memory                               */
/*****************************************************************/

{

USIGN8 *result;

result = (USIGN8 *) malloc(length);
return (result);
}



VOID memory_deallocation (ptr)
USIGN8 * ptr;

/*****************************************************************/
/*  Functionbody  memory_deallocation (ptr)                      */
/*                                                               */
/*  -->  The parameter "ptr" is a pointer to the first byte of   */
/*       memory space which will be deallocated by this function */
/*****************************************************************/

{

free((VOID *)ptr);

return;
}
```

 © 1995 PEP Modular Computers

```
/*-----------------------------------------------------------------------------
FUNCTIONAL_DESCRIPTION
This function copies a given number of Bytes (length, 1-65535) from
the source_address to the destination_address.
-----------------------------------------------------------------------------*/

VOID block_copy (source, dest, length)
     register USIGN8  *source;
     register USIGN8  *dest;
     register USIGN16 length;

{

USIGN16 l;

for (l=0;l<length;l++)
{
dest[l] = source[l] ;
}

}
```

This page has been intentionally left blank

 © 1995 PEP Modular Computers

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP