Protocol interface manual

PROFIBUS-DP Master

and PROFIBUS-DPV1 Class 1 and 2
and PROFIBUS-FDL
and MPI

Hilscher Gesellschaft für Systemautomation mbH
Rheinstraße 15
D-65795 Hattersheim
Germany

Tel. +49 (6190) 9907-0
Fax. +49 (6190) 9907-50

Hotline and Support: +49 (6190) 9907-99

Sales email: sales@hilscher.com
Hotline and Support email: hotline@hilscher.com

Web: http://www.hilscher.com

Index	Date	Version	Chapter	Revision
1	10.05.96	1.000	all	created
2	13.06.96	1.000	all	spelling correction of english translation
3	21.11.96	1.012	all	extended global status field and error numbers
4	04.12.96	1.012	all	more detailed descriptions
5	11.11.97	1.106 (CIF30-DPM) 1.004 (CIF 30-PB)	all	message definition for DPV1 expanding and Classe 2 services only available on CIF 30-PB
6	10.12.97	1.106 (CIF30-DPM) 1.004 (CIF 30-PB)	3.3	extented explanation of the global bits in the protocol status field
7	17.12.97	1.106 (CIF30-DPM) 1.004 (CIF 30-PB)	4.2.3,4.2.4, 4.2.5, 4.1.1	DDLM_Download general definition, changing of transmitter and receiver signification in some messages Extented Shared memory for CIF 30-PB
8	16.06.99	1.140 DP-Modules 1.040 PB-Modules		DPV1-Extention new Chapter: General procedure how to get the DEVICE operative without SyCon Slave parameter data set extention Master parameter data set extention explanation of Sevice Get_Cfg, new message MSAL1_Alarm_res/con correction of error numbers for Class1 READ/WRITE service
9	30.06.99			FDL_Send_Data_Reply inserted
10	02.03.00	1.148 DP-Modules 1.048 PB-Modules		-correction of DDLM_Get_Cfg message definition, -correction of hexadecimal example of slave parameter data set -new Redundant logic on PB devices -new chapter technical characteristics -configuration of Identnumber via dual-port parameter area -new parameter HighPriority-Handshake -new chapter DBM32LL
11	27.09.00	1.050		- new chapter SDA,SDN,ReplyUpdate command messages - documentation of Life_List message
12	18.10.00	1.051		- new chapter MPI read and write DB and MPI disconnect - correction of LifeList message
13	19.01.01	1.053		- divide SDA chapter into req/con and ind chapter
14	08.05.01	1.057 PB-Modules		- new error code CON_DL in Chapter SDA service - correction of msg.b command number of MPI_Read_Write_DB - new command MPI read IO, Memory, Counter and Timer
15	19.07.01			- correction of msg.function in DDLM_Life_List message - chapter of Redundancy enlarged
16	08.10.01	1.061		- correction of msg.a and msg.b command number in FDL_Send_Data_Reply_Ind - new chapter FDL_Send_Data_No_Ack_Ind / SDN - changing structure FDL_DATA_ACK_IND_STU - All Messages for FDL Services in own chapter - MPI max length
17	06.05.02	1.065		- unlock disable bit in parameter section inserted
18	03.07.02	1.068		- changing the parameter value DPM_FORMAT_INTEL/MOTOROLA to DPM_FORMAT_SWAP/NO_SWAP in section parameter data - changing the service messages MSAC2M_Abort_Req,Ind and MSAC2M_Close_ind to support multiple connections

19	06.08.03	general		- specifying the minumim needed initialization time during warmstart - new command FDL_Status_Reply - correction of the warmstart parameter bCyletime having 0.1msec as time basis
20	18.10.03	1.072	4.3.11 3.1	- new chapter FDL_SEND_TIME_SYNC_REQ - new status information about current master address - new status information about current Master Protocol Chip status

Although this software has been developed with great care and intensively tested, Hilscher Gesellschaft für Systemautomation mbH cannot guarantee the suitability of this software for any purpose not confirmed by us in writing.

Guarantee claims shall be limited to the right to require rectification. Liability for any damages which may have arisen from the use of this software or its documentation shall be limited to cases of intent.

We reserve the right to modify our products and their specifiactions at any time in as far as this contributes to technical progress. The version of the manual supplied with the software applies.

1 Indroduction

This manual describes the user interface of PROFIBUS-DP for the communication interfaces and the communication module. The aim of this manual is to support the integration of these devices into own applications based on device driver function or direct access into the dual-port memory.

The general mechanism for the data transfer is protocol independent and described in the 'general definitions' of the toolkit manual.

All parameters and data have basically the description LSB/MSB. This corresponds to the convention of the Microsoft-C-compiler. The storage format of word oriented send and receive process data can be parameterized.

1.1 Protocol Signification

1.1.1 PROFIBUS-DP Master CIF 30-DPM / COM-DPM / CIF 104-DPM

To manage the PROFIBUS-DP protocol 3 tasks are involved in the system. Therefore following entries for the protocol signification in the variables `TaskiName` are done:

```
Task2Name: 'PLC     '
Task3Name: 'USR_INTF'
Task7Name: 'FDL     '
```

1.1.2 PROFIBUS Combi Master CIF 30-PB, COM-PB, CIF 104-PB, CIF50-PB, CIF 60-PB

To manage the PROFIBUS Combi Master protocol (PROFIBUS-DP and PROFIBUS-FMS) 6 tasks are involved in the system. Therefore following entries for the protocol signification in the variables `TaskiName` are done:

```
Task1Name: 'ALI     '
Task2Name: 'PLC     '
Task3Name: 'USR_INTF'
Task4Name: 'FMS     '
Task6Name: 'LLI     '
Task7Name: 'FDL     '
```

1.2 The Process Data Interface

The DEVICE handles 512 bytes send and 512 bytes receive process data in the lower kbyte of the dual-port memory for the PROFIBUS-DP on the CIF 30-DPM/COM-DPM/CIF 104-DPM and 3584 bytes send and 3584 bytes for the PROFIBUS-DP on the CIF 30-PB. To exchange the data between the DEVICE and the HOST use the device driver function `DevExchangeIO()`or read and write directly into these locations.

After calling the function `DevExchangeIO()`the function decides on its own which handhake mechanism has to be used to read and write the process data in the rigth manner from and to the DEVICE. If these locations are accessed directly without using the device driver functionality, than you have to use the right handshake mechanism to ensure that the data is handed over safety and valid. See chapter 'IO Communication with a Process Image' in the 'toolkit general definitions' manual.

CIF 30-DPM / COM-DPM / CIF 104-DPM:

Parameter	Address	Description
SndPd	000h	Send Process Data (HOST ? DEVICE ? Network)
RecvPd	200h	Receive Process Data (Network ? DEVICE ? HOST)

CIF 30-PB / COM-PB / CIF 104-PB / CIF 50-PB / CIF 60-PB

Parameter	Address	Description
SndPd	000h	Send Process Data (HOST ? DEVICE ? Network)
RecvPd	E00h	Receive Process Data (Network ? DEVICE ? HOST)

2 Protocol Parameters

Some important parameters can be handed over to the DEVICE online. They have a higher priority than the static parameters in the internal FLASH memory usually configured by SyCon.
The execution of the socalled WARMSTART itself takes at minimum 2,5 seconds until the DEVICE is Ready again., while as maximum supervision time until the device is Ready 10 seconds can be used.

2.1 Using Device Driver Function to Write

To hand over these parameters use the device driver function `DevPutTaskParameter()`. For parameter `usNumber` use value 2, because the parameters must handed over to the task 2. For parameter `usSize` use value 16, to fix the length of the structure. Point the parameter `pvData` to the following structure below.

```
typedef struct DPM_PLC_PARAMETERStag {
  unsigned char   bMode;
  unsigned char   bCycleTime;
  unsigned char   bFormat;
  unsigned short  usWatchDogTime;
  unsigned char   bRedundant;
  unsigned char   bSlStateMethod;
  struct{
    unsigned char  biIdentNumberActive : 1;
    unsigned char  biHighPriorHandshake: 1;
    unsigned char  biUnlockDisable     : 1;
    unsigned char  biReserved          : 5;
  } bEnableBits;
  unsigned short usIdentNumber;
  unsigned char  abReserved[6];
} DPM_PLC_PARAMETERS;

/* values for bMode */
#define DPM_SET_MODE_BUSSYNC_DEVICE_CONTROLLED  0
#define DPM_SET_MODE_BUFFERED_DEVICE_CONTROLLED 1
#define DPM_SET_MODE_UNCONTROLLED               2
#define DPM_SET_MODE_BUFFERED_HOST_CONTROLLED   3
#define DPM_SET_MODE_BUSSYNC_HOST_CONTROLLED    4
/* values for bFormat */
#define DPM_FORMAT_SWAP    0x01
#define DPM_FORMAT_NO_SWAP 0x00
/* values for bSlaveStateMethod */
#define        DPM_SL_STATE_NOT_SYCHRONOUS 0
#define        DPM_SL_STATE_SYCHRONOUS     1
```

After setting up the structure and fixing it with `DevPutTaskParameter()`, the warmstart command must be performed with the `DevReset()` function. The most important parameter in this function `usMode` must be set up to 3 = WARM-START. After the warmstart is finished without error the new parameters are active.

2.2 Direct Write Access in Dual-port memory

First the parameters must be written down into the corresponding area of the dual-port memory. Then a warmstart command must be activated by setting the `Init` bit in the variable `DevFlags`. Then the DEVICE will set them valid (see the chapter 'initialization of the DEVICE' in the toolkit manual 'general definitions' for handle of the init procedure).

CIF 30-DPM / COM-DPM / CIF 104-DPM:

structure element	type	address	parameter
bMode	byte	6C0H	process data delivery (0,1,2,3,4)
bCycleTime	byte	6C1H	cycle time of DP bus cycle (0-25.5ms)
bFormat	byte	6C2H	storage format of word oriented process data (0,1)
usWatchDogTime	word	6C3H	HOST-supervision time in multiples of a msec.
bRedundant	byte	6C5H	NOT AVAILABLE ON DPM CARDS!!!!!!!
bSlStateMethod	byte	6C6H	defines the update method of Sl_State bit field
bEnableBits	byte	6C7H	Enable bits
usIdentNumber	word	6C8H	Identnumber in Motorola Format

CIF 30-PB / COM-PB / CIF 104-PB / CIF 50-PB / CIF 60-PB:

structure element	type	address	parameter
bMode	byte	1EC0H	process data delivery (0,1,2,3,4)
bCycleTime	byte	1EC1H	cycle time of DP bus cycle (0-25.5ms)
bFormat	byte	1EC2H	storage format of word oriented process data (0,1)
usWatchDogTime	word	1EC3H	HOST-supervision time in multiples of a msec.
bRedundant	byte	1EC5H	Enables, disables redundantic logic
bSlStateMethod	byte	1EC6H	defines the update method of Sl_State bit field
bEnableBits	byte	1EC7H	Enable bits
usIdentNumber	word	1EC8H	Identnumber in Motorola Format

Protocol parameters in area `Task2Parameter`

2.3 Explanation of the Protocol Parameters

The parameter `bMode` fixes the handshake mode of the process data.

The parameter `bCycleTime` fixes the minimum slave interval time in multiples of 100usec. the master has to wait for until it starts the next DP-process data exchange to all slaves. If the value is zero the DP data exchange cycle is done as fast a possible.

The parameter `bFormat` changes the storage format of word oriented process data from MSB/LSB to LSB/MSB convention and vice versa. Is the value set to 1 the DEVICE will swap the data.

The parameter `usWatchDogTime` fixes the time in multiples of 1msec. the DEVICE has to supervise the HOST program if it has started the HOST-watchdog functionality once.

The parameter `bRedundant` defines if the DEVICE shall enable the redundant logic. This function is only available on PB card DEVICES.

D7	D6	D5	D4	D3	D2	D1	D0
reserved						MST2	MST1
						redundant Master 2 passive	redundant Master 1 active

The parameter `bSlStateMethod` defines the update method of the `abSl_State` bit field within the DPM_DIAGNOSTICS structure defined the chapter 'Protocol States'. The update method can be either not sychronous to the handshake of the process data and is updated fully independent, so that the `abSl_State` field can be used to check if a slave is currently active or not, or the update method can be sychronous to the handshake of the process data. In this case the `abSl_State` field can be used to check if the different slaves input data of each confirmed process data handshake is valid and with this implicit if a slave is active in the network or not.

```
#define DPM_SL_STATE_NOT_SYCHRONOUS 0
#define DPM_SL_STATE_SYCHRONOUS     1
```

The parameter `bEnableBits` is a bit array and enables different further parameter values.

D7	D6	D5	D4	D3	D2	D1	D0
reserved					Unlock Disable	HighPrior Handshake	Enableident Number

enables the Identnumber configuration

enables the high priority handshake for buffered process data exchange handshake modes

disables the unlock slave mechnism when the DEVICE is set to NotRdy-state

The parameter `HighPriorHandhake` configures the handshake priority for the 2 buffered process handshake methods. Normally the real time operational system shares the processor performance between the PROFIBUS communication and the handshake for the process data and limits both in their required times if necessary so that both tasks are executed. If now the `HighPriorHandhake` bit is set the handshake of the process data for both buffered handshake methods will have always high priority against the PROFIBUS communication. This guarantees the fastest handshake ever and also no PROFIBUS communication errors which normally enlarge the handhake time, will do this any more. But one thing must be held the HOST application. Because the handshake has now the highest priority in the system, a very fast handshake can shut down the PROFIBUS communication. So the HOST application is forced not to do a handshake with less than 1millisecond between each.

The parameter `usIdentNumber` defines the ident number of the device. This parameter is only enable, if the corresponding enable bit is set in the variable `bEnableBits`. The ident number must be set in motorola format, that means MSB first and then LSB. If for example the Hilscher Ident-number 0x7505 should be configured 0x75 must be set in the MSB first byte and 0x05 is the LSB last byte.

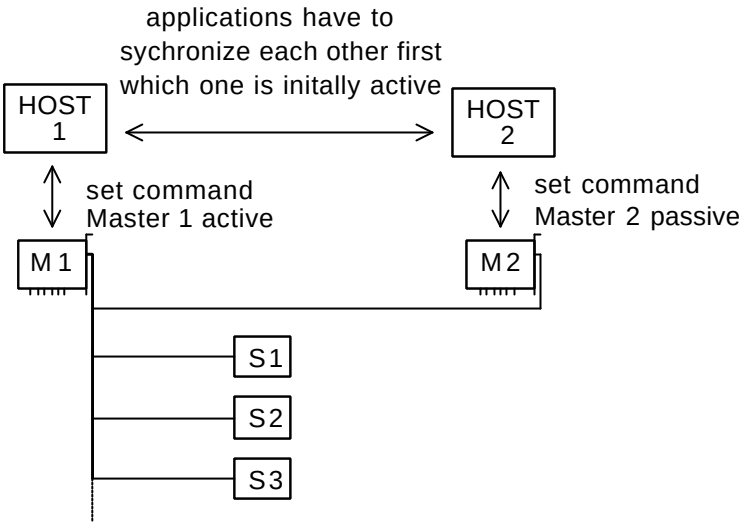
The bit `UnlockDisable` disables the Unlock Global Control command the DEVICE has to send to all assigned slaves which are driven in process data exchange state, when the `NotRdy` bit is set and the DEVICE goes to STOP mode. This prevents the slaves to set back there output data in this case.

2.3.1 The Redundant Logic

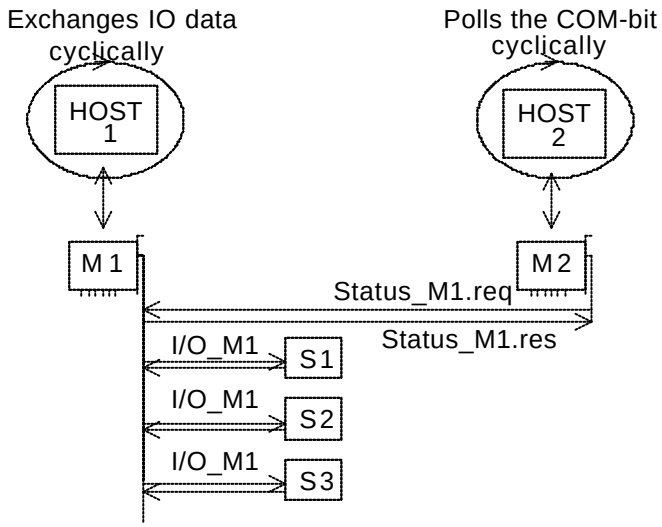
A as redundant Master 2 passive defined DEVICE will start to request the other masters existence by a special Status.req poll telegram cyclically. A master 1 active DEVICE will always behave like a standard master, but will additionally respond to the Master 2 request poll frame with Status.res.
A Master 1 active will use the PROFIBUS bus address directly which is configured in its master bus parameter data set (see chapter below) to be present in the network. A Master 2 passive DEVICE will always use the configured bus address - 1 to be present in the network first time. If the master address is configured to address 0, the passive master will automatically adjust its address to 125.
As soon as the Master 2 passive DEVICE will receive a valid response from Master 1 DEVICE, it will set the bit `COM` in the cell `bHostFlags`. If the Master 1 active do not respond to the Master 2 requests any more, the Master 2 will indicate this event by clearing the bit `COM`.
The HOST application of the passive Master 2 has to poll the `COM` bit all the time. If it sees the `COM` bit gone, it has to inform the other Master 1 application about this. The HOST application of Master 1 usally configures then its master to be a passive master though a new warmstart command or just by setting the `NotRdy` in the cell `bDevFlags`. The Master 1 then immediatly stops its network activity by neeing a passive Master 2 device and remains in the passive state and polls for an active Master 1. In parallel the HOST application of the inital passive Master 2 has to clear the bit `NotRdy` in the Master 2 DEVICE to enable its network activity by being a Master 1 DEVICE now. It will reconfigure its bus address to the standard value and start to communicate with the configured slave devices without reinitializing them. This suppresses that the output values of the slaves go into the save zero condition.

The HOST application of the active Master 1 should use the watchdog trigger mechnism of its master card. If the HOST application dies, then the master 1 DEVICE recognizes this and stops immediatly all its PROFIBUS communication and goes bus off. Then the active Master 1 is not present any longer i the view of PROFIBUS and the other passive Master 2 is able to gain the bus access to the slave devices. The watchdog timer can be set up either in the confguration tool of the master or is part of the protocol parameter which are explained in this chapter.

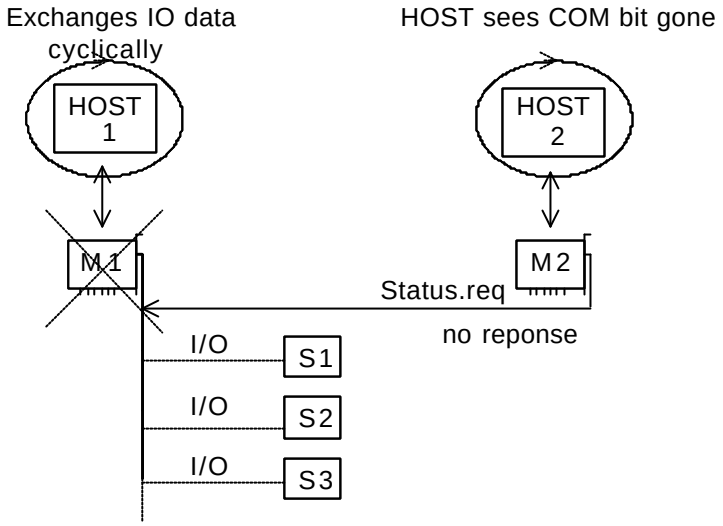
Both HOSTs have to sychronize themself first before starting the DEVICEs. It must be clear which HOST application is initially active and which one is initially passive. For that reason both applications need a separate connection (via Ethernet for example) next to the PROFIBUS line to handle this sychronization.



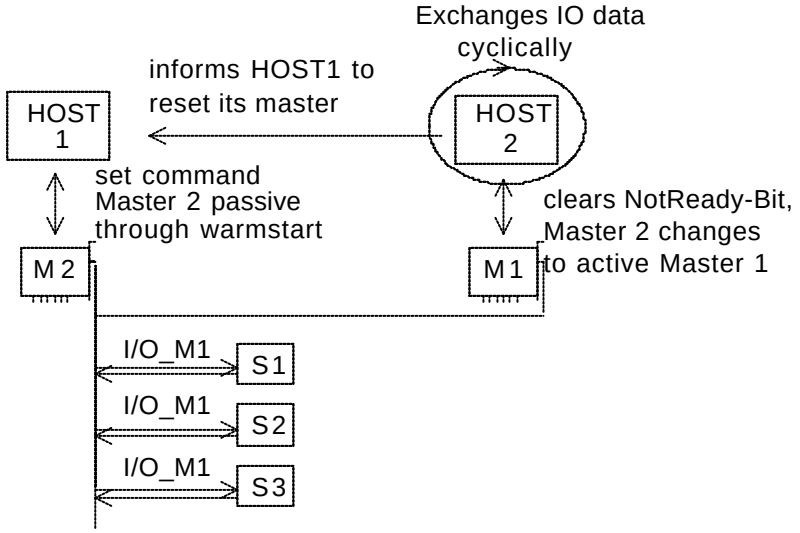
Once the DEVICEs are started, the active Master 1 requests cyclically the assigned slaves and the passive Master 2 polls the Master 1 station, if it is still present. The APP 2 of the passive Master has to poll the `COM` bit in the cell `bHostFlags`, which relects the current status of the M 1 station.



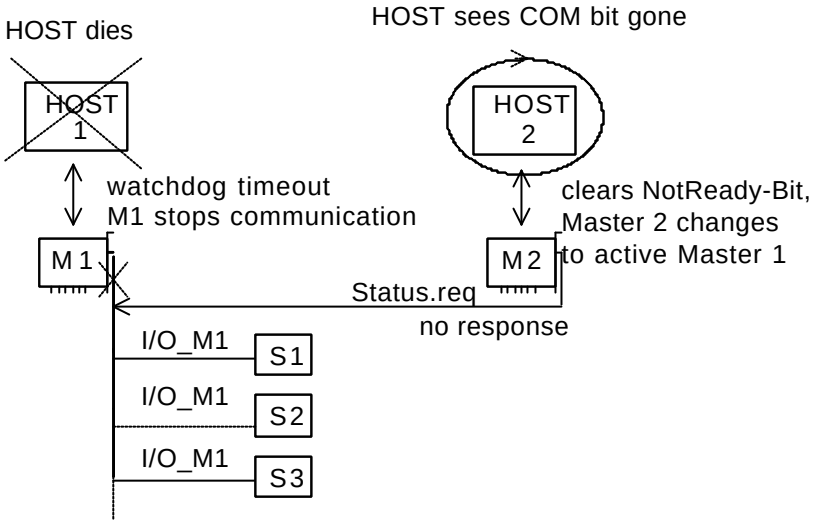
If the active Master 1 now dies, the status requests of the passive master 2 are not confirmed by the Master 1 any longer and the HOST in the Master 2 sees the `COM` bit in the cell `bHostFlags` gone. The IO exchange to the slave devices is stopped in that moment the master dies, but the watchdog functionality keeps them alive for a short time.(this time depends on the settings in the bus parameter of the master 1).



After seen the `COM` bit gone to logical 0, the HOST 2 application informs the HOST 1 application to change its master from active to passive by a new warmstart-reset. The parameter `bRedundant` must be set to value 02H in his case. Without any delay and given the information to the HOST 1 application, the HOST 2 clears the `NotReady`-bit in the cell `bDevFlags`. The Master 2 passive changes immediatly to Master 1 active (and changes its own PROFIBUS address) and starts to poll the slaves without restarting them via PROFIBUS command. The switchover time between the old Master 1 gets lost and clearing the `NotReady`-bit must be less or equal the watchdog time of the slave devices. Else the slaves will detect a timeout and clear their output data automatically. Time measurements at 12MBaud show a minimum reachable switchover timer of 1msec at fast HOST applications.



Another cenario that can happen is that the HOST application of the active Master 1 dies. To prevent that the active Master 1 then is still running without having valid process data and no valid HOST application running above him, the HOST application should support the Watchdog-Trigger mechanism, which is explained in the Toolkit General Definitions manual.
The Master 1 then detects that the HOST application timed out and stops immediatly all network acitivity. This gives the passive Master 2 then the chance to gain the bus access to the PROFIBUS slaves. The passive Master 2 clears the `COM` bit in the cell `bHostFlags` and the HOST 2 application has to clear the `NotReady` bit. Then this master changes to from passive to active and polls the IO slaves.



2.3.1.1 The best way to perfom the redundant switch over

So the settings for the protocol parameter in the dual-port memory must be for the active card : `bRedundant` = 0x01 = MST1 bit set
passive card: `bRedundant` = 0x02 = MST2 bit set

This is the only difference between both DEVICES, nothing else must be changed and the whole configuration of the PROFIBUS slave data sets etc. is all the same in both card. If the cards are configured with the SyCon configuration tool both cards get exactly the same PROFIBUS configuration download file during the download sequence.

ONE MAIN THING IS VERY IMPORTANT! : Avoid that both cards have the same profibus address during the switchover. Remember both cards get the same PROFIBUS address with SyCon configuration tool, then you perform a warmstart command to enable the parameter `bRedundant`. Decide first of all in both HOST applications which DEVICE shall be the passive card first and which shall be the active card. Usually at the very first startup, one card builds always the passive card and the other is then the active card.

You have to perfom the warmstart command to the intended passive DEVICE first if the decision is done. The DEVICE then automatically adjust the configured PROFIBUS address to its PROFIBUS address minus one. So if you have configured it to 1 with Sycon the passive DEVICE will have the address 0 automatically for example. Then perform the warmstart to the active DEVICE, the active DEVICE will use the SyCon configured address = 1 without changing it.
Now if you would switch over with the passive DEVICE by clearing the `NotReady` bit in the cell `bDevFlags` and the passive DEVICE will then change automatically the profibus address back to value 1 to be active and the active DEVICE is still active, then there is a conflict and then the passive DEVICE sees the old active DEVICE still active and the PROFIBUS master chip of the passive DEVICE goes to bus off and stops its work. The passive card get useless at this moment and can't fulfill its purpose. This is the problem.
For that reason both HOST application has to sychronize themselve during the switchover. The best way is during switchover is the following: first change the parameter of the initially active DEVICE to bRedundancy = MST2 = 0x02 = passive and start immediatly afterwards a warmstart command by writing the `Init` bit in the cell `bDevFlags`. After that the DEVICE of course goes immediatly into reset and its PROFIBUS master chip goes bus off for at least one second. This is enough time then to start the passive card on the other hand to ensure the PROFIBUS communication to all slaves again. Now the active card is not present any more in the PROFIBUS network for a while and the other HOST application has to clear now the `NotReady` bit in the cell `bDevFlags` of the passive DEVICE and the passive DEVICE changes to active DEVICE right away without any problem, because the old active card is still in reset state. This changed passive card now replaces now the missing PROFIBUS process data exchange of the old active DEVICE to all the slaves. After 1 to 2 seconds the old active DEVICE comes back from its initializations, but has changed to the passive mode because of given parameter `bRedundant` = MST2 = 0x02 and remains now passive. This is a save procedure to change from passive to active and from active to passive.

If you would change active to passive and vice versa with `NotReady` bit clearing and setting in both DEVICEs , the problem is that the switchover time of both HOST application to set or to clear the bit isn't 100% stable normally and then one DEVICE could be passive ealier then the other goes from active to passive and so two PROFIBUS addresses are present in the network and a conflict exists. In 10 cases of switchover at least 1 case do not succeed because of that reason and you lose the control of your PROFIBUS system.

3 Protocol States

The protocol states built by the DEVICE form the diagnostic interface between the HOST and the PROFIBUS-DP.
The established structure informs about global bus states as well as individual states of the managed slave stations. To hold the information preferably compact, the slave specific information are held in state bit fields.
The first 4 state variables in the structure inform about global master and network state informations. They are followed by a bus error event counter. After this field an unused reserved area of 26 byte is following. The following first 16 bytes characterize each slave as configured and handled. The next 16 bytes characterizes each slave station as active or inactive in the network, followed by 16 bytes which serve to refer the diagnostic bit of every slave station.

3.1 Using Device Driver Funtions

Use the device driver function `DevGetTaskState()` to read the states. For parameter `usNumber` use value 2, because the parameter must be read from the task state area of the task 2. For parameter `usSize` use value 64, which is the length of the structure below. Point `pvData` to the following structure.

```
typedef struct DPM_DIAGNOSTICStag {
  struct
  {
    unsigned char bCtrl     : 1;
    unsigned char bAClr     : 1;
    unsigned char bNonExch  : 1;
    unsigned char bFatal    : 1;
    unsigned char bEvent    : 1;
    unsigned char bNRdy     : 1;
    unsigned char bTout     : 1;
    unisgned char bReserved : 1;
  } bGlobalBits;

  unsigned char   bDPM_State;

  struct
  {
    unsigned char bErr_rem_adr;
    unsigned char bErr_event;
  } tError;

  unsigned short  usBus_error_cnt;
  unsigned short  usTime_out_cnt;
  unsigned char   bCurMstAdr;
  unsigned char   bOnline;
  unsigned char   abReserved[6];

  unsigned char   abSl_cfg[16];
  unsigned char   abSl_state[16];
  unsigned char   abSl_diag[16];
} DPM_DIAGNOSTICS;
```

3.2 Direct Read Access in Dual-port memory

Read the bus state structure directly from the following dual-port memory location:

variable	type	address CIF 30-DPM	address CIF 30-PB	short signification
Global_bits	1 byte	740H	1F40H	global error bits
DPM_state	1 byte	741H	1F41H	main state of the master system
Err_rem_adr	1 byte	742H	1F42H	faulty remote address
Err_event	1 byte	743H	1F43H	error number
Bus_error_cnt	2 bytes	744H	1F44H	heavy bus error counter
Time_out_cnt	2 Bytes	746H	1F46H	number of rejected PROFIBUS-Telegrams
bCurMstAdr	1 byte	748H	1F48H	Current Profibus Master Address
bOnline	1 byte	749H	1F49H	Variable reflecting the current status of the Master Protocol Chip
reserved	6 bytes	746H-74FH	1F46H-1F4FH	reserved error variables
Sl_cfg	16 bytes	750H-75FH	1F50H-1F5FH	see the table below
Sl_state	16 bytes	760H-76FH	1F60H-1F6FH	see the table below
Sl_diag	16 Bytes	770H-77FH	1F70H-1F7FH	see the table below

Global bus state field in area `Task2State`

3.3 Explanation of the Protocol States

- Global_bits

D7	D6	D5	D4	D3	D2	D1	D0
0	TOUT	NRDY	EVE	FAT	NEXC	ACLR	CTRL

CONTROL-ERROR:
parameterization error

AUTO-CLEAR-ERROR:
DEVICE stopped the communication to all slaves and reached the auto-clear end state

NON-EXCHANGE-ERROR
At least one slave has not reached the data exchange state and no process data are exchange with.

FATAL-ERROR:
Because of heavy bus error, no further bus communication is possible

EVENT-ERROR:
The DEVICE has detected bus short circuits. The number of detected events are fixed in the Bus_error_cnt variable. The bit will be set when the first event was detected and will not be deleted any more.

HOST-NOT-READY-NOTIFICATION:
Indicates if the HOST program has set its state to operative or not. If the bit is set the HOST program ist not ready to communicate

TIMEOUT-ERROR:
The DEVICE has detected an overstepped timout supervision time because of rejected PROFIBUS telegramms. It's an indication for bus short circuits while the master interrupts the communication.The number of detected timeouts are fixed in the Time_out_cnt variable.The bit will be set when the first timeout was detected and will not be deleted any more.

reserved

The bit field serves as collective display of global notifications. Notified errors can either occur at the DEVICE itself or at the slaves. To distinguish the different errors the variable `err_rem_adr` contains the error location (bus address), while the variable `err_event` is fixing the corresponding error number. If more than one error is determined, the error location will always show the lowest faulty bus address.

- Variable DPM_state

This variable represents the main state of the master system. Following values are possible:

00H: state OFFLINE
40H: state STOP
80H: state CLEAR
C0H: state OPERATE

- Variable Err_rem_adr

Some bits in the `Global_Bit` field indicating errors in the network or in the DEVICE itself have always a closer error desciption. In these cases the variable `Err_rem_adr` represents the source of the error. The source where the error was detected can be either the DEVICE itself., then the variable contains the value 255, or the error was detected at or reported by a network device. Then the variable is filled up with this station address directly and has a range from 0 to 125.

- Variable Err_event

To complete the error description the variable `Err_event` delivers next to the error source the corresponding error number. All possible numbers are listed below.

- Variable Bus_error_cnt

In this variable counts heavy bus errors, for example bus short circuits.

- Variable Time_out_cnt

This variable contains the number of rejected PROFIBUS telegrams because of heavy bus error.

- Variable bCurMstAdr

This variable contains the current active Master address. The status is also updated in case of using the Redundant logic at the active and passive Master.

- Variable bOnline

This variable reflects the current status of the Master Protokoll-Chip. If the value i 0 then the Chip is in Offline State, if the value is 1 the Chip is on Online State. The variable is updated cyclically.

- Variable reserved

This data block is reserved.

- Variable Sl_cfg

This variable is a field of 16 bytes and contains the paremeterization state of each slave station. The following table shows, which bit is related to which slave station address:

Bit / Offset	D7	D6	D5	D4	D3	D2	D1	D0
750H / 1F50H	7	6	5	4	3	2	1	0
751H / 1F51H	15	14	13	12	11	10	9	8
752H / 1F52H	23	22	21	20	19	18	17	16
...								
75FH / 1F5FH	127	126	125	124	123	122	121	120

Table of the relation between node address and the Sl_Cfg bit

If the Sl_cfg bit of the corresponding slave is logical
'1', the slave is configured in the master, and serviced in its states.
'0', the slave is not configured in the master.

- Variable Sl_state

This variable is a field of 16 bytes and contains the state of each slave station. The following table shows, which bit is related to which slave station address:

Bit / Offset	D7	D6	D5	D4	D3	D2	D1	D0
760H / 1F60H	7	6	5	4	3	2	1	0
761H / 1F61H	15	14	13	12	11	10	9	8
762H / 1F62H	23	22	21	20	19	18	17	16
...								
76FH / 1F6FH	127	126	125	124	123	122	121	120

Table of the relation between slave station address and the Sl_state bit

If the Sl_state bit of the corresponding slave station is logical
'1', the slave and the master are exchanging their I/O data.
'0', the slave and the master are not exchanging their I/O data.

The values in variable Sl_state are only valid, if the master runs the main state OPERATE.

- Variable Sl_diag

This variable is a field of 16 bytes containg the diagnostic bit of each slave. The following table shows the relationship between the slave station address and the corresponding bit in the variable Sl_diag.

Bit / Offset	D7	D6	D5	D4	D3	D2	D1	D0
770H / 1F70H	7	6	5	4	3	2	1	0
771H / 1F71H	15	14	13	12	11	10	9	8

772H / 1F72H	23	22	21	20	19	18	17	16
...								
77FH / 1F7FH	127	126	125	124	123	122	121	120

Table of the relationship between slave station address and the Sl_diag variable

If the Sl_diag bit of the corresponding slave station is logical

'1', latest received slave diagnostic data are available in the internal diagnostic buffer. This data can be read by the user with a message which is described in the chapter 'The message interface' in this manual.

'0', since the last diagnostic buffer read access of the HOST, no values were change in this buffer.

The values in variable Sl_diag are only valid, if the master runs the main state OPERATE.

	sl_state = 0	sl_state = 1
sl_diag = 0	- No DataIOExchange between master and slave. - Perhaps this slave is not configured or not responsive.	- Slave is present on the bus. - DataIOExchange between master and slave.
sl_diag = 1	- The master and the corresponding slave do not exchange their I/O data. - The master holds newly received diagnostic data in the internal diagnostic buffer.	- Slave is present on the bus. - The master and the corresponding slave do exchange their I/O data. - The master holds newly received diagnostic data in the internal diagnostic buffer.

Relationchip between Sl_state bit and Sl_diag bit

The following error numbers are valid for `Err_event`, if `Err_rem_adr` is 255:

err_event	signification	error source	help
0	no mistakes appear		
50	USR_INTF-Task not found	DEVICE	contact technical support
51	no global data-field	DEVICE	contact technical support
52	FDL-Task not found	DEVICE	contact technical support
53	PLC-Task not found	DEVICE	contact technical support
54	non existing master parameters	DEVICE	execute download of data base again
55	faulty parameter-value in the master parameters	project planning	contact technical support
56	non existing slave parameters	project planning	execute download of data base again
57	faulty parameter-value in a slave parameters datafile	project planning	contact technical support
58	double slave address	project planning	check projected addresses
59	projected send process data offset address of a participant outside the allowable border of 0- 255	project planning	check projected addresses
60	projected receive process data offset address of a participant outside the allowable border of 0- 255	project planning	check projected addresses
61	data-areas of slaves are overlapping in the send process data	project planning	check projected addresses
62	data-areas of slaves overlapping in the receive process data	project planning	check projected addresses
63	unknown process data handshake	warmstart	check warmstart parameters
64	free RAM exeeded	DEVICE	contact technical support
65	faulty slave parameter data sets	project planning	contact technical support
202	no segment for the treatment free	DEVICE	contact technical support
212	faulty reading of a data base	DEVICE	execute download of data base again
213	structure-surrender to operating system faulty	DEVICE	contact technical support
220	Software Watchdog error	HOST	check HOST program

221	No Data Acknowlegde in process data handshake mode 0	HOST	HOST program hasn't acknowledge the last handshake in time
222	Master in Auto_Clear	Slave Device	the auto_clear mode was activated, because one slave is missing during runtime
225	No further Segments	DEVICE	contact technical support

The following error numbers are valid for `Err_event`, if `Err_rem_adr` is unequal 255:

err_event	signification	error source	help
2	station reports overflow	master telegram	check length of configured slave configuration or parameter data.
3	request function of master is not activated in the station	master telegram	check slave if PROFIBUS-DP norm compatible
9	no answer-data, although the slave must reponse with data	slave	check configuration data of the station and compare it with the physical I/O data length
17	no response of the station	slave	check bus cable, check bus address of slave
18	master not into the logical token ring	DEVICE	check FDL-Address of master or highest-station-Address of other master systems. examine bus cableing to bus short circuits.
21	faulty parameter in request	master telegram	contact hotline

4 The Message Interface

The following send and receive messages are exchanged with the DEVICE via its mailboxes in the structure like it is described in the chapter 'definition of the message interface' in the toolkit manual.

To put and get messages to respectively from the DEVICE through its mailboxes use the device driver functions `DevPutMessage()` or `DevGetMessage()`. With direct access to the dual-port memory you must write the message in the `DevMailbox` or read the message out of the `HostMailbox` with the mechanism described in the toolkit manual.

The structures of this messages and its values are described in the sections below.

4.1 The PLC-Task

The PLC-Task manages the process input and output data and handle the steering of the DP cycles corresponding the parameterization. Therefore the task communicates with the USR_INTF-Task and starts the DP cycle according to the parametrized operation mode. The task has implemented the following functions:

- activates DP cycles.
- mapping of the physical addresses of the data to the logical addresses of the data in the dual-port memory.

The task manages following message command:

DPM_Shared_Memory	write consistent data block into the send process data `SndPd` during one DP cycle or read consistent data block from `RecvPd` during one DP cycle .

4.1.1 DPM_Shared_Memory

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	2	receiver = PLC-TASK
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	 8 8+m	length of the message read access write access
	msg.nr	byte	j	number of message (optional)
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	17	command = DPM_Shared_Memory
	msg.e	byte	0	unused
Extended message header	msg.device_adr	byte	0	device address, unused
	msg.data_area	byte	 0 1 2	data area: msg.function decides the data area receive process data area send process data area
	msg.data_adr	word	 0-255,1791 0-511,3583 0-255,1791	address offset refer to the data type word-offset address byte-offset address word-offset address if bit access
	msg.data_idx	byte	0-15	bit position within the word offset address if bit access
	msg.data_cnt	byte	m	count of read or write data referring to the data type
	msg.data_type	byte	 6 5,10 14	data type: TASK_TDT_UINT16: word TASK_TDT_UINT8: octet-string TASK_TDT_BIT: bit
	msg.function	byte	 1 2	function: TASK_TFC_READ = read access TASK_TFC_WRITE = write access
Data	msg.d[0]	byte	x	write access: first data to be written read access: unused
	...	...	...	...
	msg.d[m-1]	byte	z	write access: last data to be written read access: unused

The command serves the user program to write data of a definite length into the send process data buffer or to read from the receive process data. The command can be used in all process data handshake modes.
With the command the data types word, bytes or bits can be selected. The firmware of the DEVICE guarantees that the read or write access of the data will be done safely during two active DP cycles.
This command is an other possibility to get access to the process data in the dual-port memory. Its disadvantage is the slower access then the direct reading or writing the dual-port memory. But the main advantage is that you have the access to

the process data also via a message, when an old driver for example has already message functionality in it. We use this message in all diagnostic tools too.

The data type is fixed in the byte `msg.data_type`. Only the values decimal 6 for words, 5 or 10 for byte strings and 14 for bits are allowed.

The read access is distinguished from the write access in the byte `msg.function`. A 1 is valid for read access and 2 for write access.

The data area to be read from, is fixed in `msg.data_area`. 1 is valid for the receive process data buffer and 2 for the send process data buffer. In case of the value 0, the value placed in `msg.function` decides the data area automatically.

The count of the data to be read or to be written is fixed by the value of `msg.data_cnt`. The count refers to the chosen data type. Maximum permitted values are 119 for words, 239 for byte and 255 for bits.

The offset address is fixed in the word `msg.data_adr`. The specified address must be refered on the chosen data type and is interpreted from the DEVICE as the relative address to the start address in the send process data or the receive process data. The maximum values are decimal 255 for word, 511 or byte and 255 for bit access if CIF 104, COM and CIF 30-DPM card is used or 1791 for word, 3583 or byte and 1791 if CIF 30-PB is used. In case of bit access the value in `msg.data_idx` additionally fixes the relative offset in the word to be read or to be written. For the other accesses the value doesn't have any meaning.

The data at `msg.d[]` are unused, if read access is chosen, while in write access in this area the send data must be written in. Words must be written in Intel format - LSB before MSB - and bits must be put in there in packed form. For example to write 5 bits, the first data byte `msg.d[0]` must be placed in the bits 0-4 to be valid.

answer message to the user			
variable	type	value	signification
msg.rx	byte	16	receiver = user at HOST
msg.tx	byte	2	transmitter = PLC-Task
msg.ln	byte	 8+m 8 0	length of the message read access write access error
msg.nr	byte	j	number of message
msg.a	byte	17	answer = DPM_Shared_Memory
msg.f	byte	0 f	no error error nummber see following table
msg.b	byte	0	no command
msg.e	byte	0	unused
msg.device_adr	byte	0	device address, unused
msg.data_area	byte	 0 1 2	data area: datafnc decides the data area receive process data area send process data area
msg.data_adr	word	 0-255 0-511 0-255	address offset refer to the data type word-offset address byte-offset address word-offset address if bit access
msg.data_idx	byte	0-15	bit position within the word offset address if bit access
msg.data_cnt	byte	m	count of read or write data referring to the data type
msg.data_type	byte	 6 5,10 14	data type: TASK_TDT_UINT16: word TASK_TDT_UINT8: octet-string TASK_TDT_BIT: bit
msg.function	byte	 1 2	function: TASK_TFC_READ = read access TASK_TFC_WRITE = write access
msg.d[0]	byte	x	read access: first data be read write access: unused
...	...	...	...
msg.d[m-1]		z	read access: last data be read write access: unused

In the answer message the `msg.f` byte reports back the information, if the command could be executed. If the error byte is 0, then the service could be finished without any error.

The extended telegram header of the answer message is not be changed if the command could be executed correctly, so that the user can assign the answer message unequivocally to the sent command message.

The convention for the data `msg.d[]` comming back in the answer message is the same one as the convention for the command messages. That means that words will be placed in Intel format and bits are placed in packed format.

The byte `msg.f` in the answer message can have the following values:

error number msg.f	signification
0	no error, command executed
162	illegal address area
163	data address together with the data count results an overflow in the buffer length
165	illegal data count
166	unknown data type
167	unknown function

4.2 The USR_INTF-Task

The USR_INTF-Task includes the two functional blocks:

- sceduler
- slave handler

The sceduler manages the global system steering of the PROFIBUS-DP. It can be divided in the mainstates OFFLINE, STOP, CLEAR and OPERATE.
For every slave station one further state machine is implemented. These indiviual state machines named slave handler are managing the several states of each slave like getting diagnostic, doing parameterization, doing configuration and doing the process data exchange.

The USR_INTF-Task uses following messages:

DDLM_Start_Seq	start multiplexed download of one data set file
DDLM_End_Seq	end multiplxed download of one data set file
DDLM_Download	download of data set file
DDLM_Upload	upload of date set file
DDLM_Global_Control	send DP command to one or many slave
DDLM_Slave_Diag	read internal saved diagnostic data of one slave
DDLM_Set_Prm	transfer new parameter values to a specific slave

The USR_INTF-Task needs the configuration data for the PROFIBUS-DP.

Normally the configuration will be downloaded by the SyConPB configuration tool statically into the FLASH memory. The task reads out this data block when the DEVICE starts up. If all parameters are valid the task starts its slave handlers and goes into the mode OPERATE.

If no static download of the configuration data is wished, all these data can be handed over online to the DEVICE by a message download from the HOST program. But before doing this, you have to prevent the DEVICE to start up with possible downloaded static parameters. This can be done by deleting the data base 'PROFIBUS' with the ComPro tool before. Then the DEVICE starts up without finding any configuration data file and the online message download can be done by the HOST program like it is described in the following chapters.
NOTE! If no data base exists on the DEVICE, the DEVICE must be initialized with protocol parameters (see chapter: protocol parameters) before the message download is done, to fix the process data handshake mode and the storage format.

ANNEX: We recommend to get the PROFIBUS norm specification named EN 50170 (DIN 19245-T3,T1) when the online download of the parameters is used. The configuration data containments of the messages have exactly the same structure and meaning like it is described in the norm specification.

4.2.1 Starting and Stopping Communication during Runtime

4.2.1.1 Using Device Driver Function to Write

Use the function `DevSetHostState()` together with the parameter `HOST_NOT_READY` to stop the network communication. Use the parameter `HOST_READY` to start or restart the communication.

4.2.1.2 Direct Write Access in Dual-Port

To start and stop the communication of the DEVICE you have to clear and set the bit `NotRdy` in the cell `bDevFlags`. Clearing the bit will start the network communication while setting the bit stops the communication.

ATTENTION: Stopping the communication will always cause a reset of the network modules output data.

4.2.2 Deleting Existing Data base in the DEVICE

Normally the configuration will be downloaded by the SyCon configuration tool statically into the FLASH memory. The DEVICE reads out this data block during its startup. If all parameters are valid the DEVICE starts its slave handlers and goes into the mode OPERATE. Then the message download procedure like it is described in the chapters below can not be used any more.

If no static download of the configuration data is wished, all these data can be handed over online to the DEVICE by a message download from the HOST program using the functions Start,End and Download. But before doing this, you have to prevent the DEVICE to start up with possible downloaded static parameters. This can be done by deleting the data base by message service before. Then the DEVICE starts up without finding any configuration data base and then the online message download can be proceeded by the HOST program like it is described in the following chapters.
IMPORTANT NOTE! If no data base exists within the DEVICE, the DEVICE must be initialized with protocol parameters (see chapter: protocol parameters) before the message download is done, to fix the process data handshake mode and the storage format etc.

command message			
variable	type	value	signification
msg.rx	byte	0	receiver = RCS-Task
msg.tx	byte	16	transmitter = user at HOST
msg.ln	byte	2	length of the message
msg.nr	byte	j	number of the message
msg.a	byte	0	no answer
msg.f	byte	0	no error
msg.b	byte	6	command = data base access
msg.e	byte	0	extention, not used
msg.d[0]	Byte	4	mode = delete data base
msg.d[1]	Byte	8 0	startsegment of the data base CIF30-DPM,CIF104-DPM, COM-DPM,COM-PB, CIF104-PB,CIF 50-PB CIF 30-PB,CIF 60-PB

answer message			
variable	type	value	signification
msg.rx	byte	16	receiver = user at HOST
msg.tx	byte	0	transmitter = RCS-Task
msg.ln	byte	1	length of message
msg.nr	byte	j	number of the message
msg.a	byte	6	answer = data base access
msg.f	byte	f	error, state
msg.b	byte	0	no command
msg.e	byte	0	extension

The time for deleting the data base depents on the used FLASH memory, so sending back the answer message can take up to 3 seconds.

4.2.3 DDLM Up-/Download - Configuring the Master Online

4.2.3.1 DDLM_Start_Seq

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	4	length of the message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	67	command = DDLM_Start_Seq
	msg.e	byte	0	extension
DDLM_START_SEQ_REQUEST	msg.d[0]	Byte	0	Req_Adr, unused
	msg.d[1]	Byte	0-125 255	Area_Code, DP-slave-parameter file local access protection deactivated
	msg.d[2]	Word	0-65535	Timeout, mutiple of 1 ms

The command starts a blocked download in the stated `Area_Code` for one download data set file. To complete the download the command `DDLM_End_Seq` must be called after finishing the download sequence for each downloaded data set file.

answer message			
variable	type	value	signification
msg.rx	byte	16	receiver = user at HOST
msg.tx	byte	3	transmitter = USR_INTF-Task
msg.ln	byte	1	length of message
msg.nr	byte	j	number of the message
msg.a	byte	67	answer = DDLM_Start_Seq
msg.f	byte	f	error, state
msg.b	byte	0	no command
msg.e	byte	k	extension
msg.d[0]	byte	240	Max_Len_Data_Unit

The value `Max_Len_Data_Unit` fixes the maximum length of the parameter `Data` per `DDLM_Download` message.

Possible values for `msg.f` are the following:

error number msg.f	signification
0	no error
52	CON_NI, Area_Code unknown

See below the corresponding structures in the header file:

DDLM_START_SEQ_REQUEST
DDLM_START_SEQ_CONFIRM

4.2.3.2 DDLM_End_Seq

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	1	length of the message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	69	command = DDLM_End_Seq
	msg.e	byte	0	extension
DDLM_END_SEQ_REQUEST	msg.d[0]	byte	0	Req_Adr, unused

The command ends the blocked download of one data set file and activates the data check and the data itself then, if no error could be located.

answer message			
variable	type	value	signification
msg.rx	byte	16	receiver = user at HOST
msg.tx	byte	3	transmitter = USR_INTF-Task
msg.ln	byte	0	length of message
msg.nr	byte	j	number of the message
msg.a	byte	69	answer = DDLM_End_Seq
msg.f	byte	f	error, state
msg.b	byte	0	no command
msg.e	byte	k	extension

Possible values for `msg.f` are the following :

error number msg.f	signification
0	no error
59	CON_DI, data incomplete or faulty

See below the corresponding structure in the header file:

DDLM_END_SEQ_REQUEST

4.2.3.3 DDLM_Download

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	4 + x	length of the message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	68	command = DDLM_Download
	msg.e	byte	0	extention, not used
DDLM_DOWNLOAD_REQUEST	msg.d[0]	byte	0	Req_Adr, unused
	msg.d[1]	byte	0-125 127	Area_Code, DP-slave parameter file master bus parameters
	msg.d[2]	word	0-760	Add_Offset
	msg.d[4... x+3]	byte string		Data[240]

This command allows to hand over the master bus parameters or the slave parameter data files. This is recommended, if no static downloaded data base exists within the DEVICE and the parameterization should happen from the HOST application without using SyCon tool for configuration.

In accordance to the PROFIBUS DP norm description two ways to download the different data files have been implemented. A data file can be downloaded either in one call (single download) or if it is to large (larger then 240 bytes that would fit in one message) in block calls (sequenced download) into an internal download area (length 1000 bytes). After the download cycle is finished completly the specified data is checked and copied afterwards into the DEVICE access area. Then the next download can be started into the now freed download area.

The parameter `Area_Code` fixes the destination area (master parameter or slave parameter file). The start offset in the download area where the data will be written into is fixed in the variable `Add_Offset`.
If a data file shall be transferred sequenced, the command `DDLM_Start_Seq` must be activated before initializing the download sequence. The sequence will be finished after the command `DDLM_End_Seq` is called. Even then the parameters will be checked and be set valid if no error is recognized.

See below the corresponding structure in the header file:

DDLM_DOWNLOAD_REQUEST

answer message			
variable	type	value	signification
msg.rx	byte	16	receiver = user at HOST
msg.tx	byte	3	transmitter = USR_INTF-Task
msg.ln	byte	0	length of message
msg.nr	byte	j	number of the message
msg.a	byte	68	answer = DDLM_Download
msg.f	byte	f	error, state
msg.b	byte	0	no command
msg.e	byte	0	extention, not used

Possible values for `msg.f` are the following:

error number msg.f	signification
0	no error
19	CON_NO, master not in offline configuration mode. Delete the data base within the DEVICE, else the DEVICE is not able to execute the download of the dataset file
21	CON_IV, parameter fault in request. Please check the downloaded parameter in your request message. The Siemens master chip has denied the bus parameter data set.
48	CON_TO, timeout. If Start and End_Seq messaging is used a message timeout occured during the sequenced download.
52	CON_NI, area_code unknown. Please check the area code of the message in its limits.
53	CON_EA, overstep of the maximum buffer length of 1000 bytes per dataset file.
55	CON_IP, faulty parameter detected. The configured data_control time is set to zero. This is not allowed
57	CON_SE, sequence error. End_seq was called without Start_seq or the area_code is different in the download message in comparison to the Start_seq area_code
59	CON_DI, data incomplete or faulty. The check routine for the downloaded parameter file detected inconsistencies

4.2.3.4 DDLM_Upload

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	5	length of the message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	80	command = DDLM_Upload
	msg.e	byte	0	extention, not used
DDLM_UPLOAD_REQUEST	msg.d[0]	byte	0	Req_Adr, unused
	msg.d[1]	byte	0-125 127	Area_Code, DP-slave parameter file master bus parameters
	msg.d[2..3]	word	0-760	Add_Offset
	msg.d[4]	byte	1-240	Data_Len

This command allows to read back the master bus parameters or the slave parameter data files.

The difference to the `DDLM_Download` command is, that the functions `DDLM_Start_Seq` and `DDLM_End_Seq` can not be used, but the data consistency is guaranteed anyway, but not supervised via the programmable timer algorithm.

The parameter `Area_Code` fixes the source area and can be either the master bus parameter file = 127 or a slave parameter file = 0-125. The start offset within the corresponding upload area where the data will be read from must be fixed in the variable `Add_Offset`. The number of data bytes that shall be read in maximum with the service have to be inserted in the `Data_Len` variable.
If the variable `Add_Offset` plus `Data_Len` overstep the maximum actual loaded file length then the DEVICE will return no error, but return all the rest data beginning at `Add_Offset` position.

See below the corresponding structure in the header file:

DDLM_UPLOAD_REQUEST

answer message			
variable	type	value	signification
msg.rx	byte	16	receiver = user at HOST
msg.tx	byte	3	transmitter = USR_INTF-Task
msg.ln	byte	x	length of message
msg.nr	byte	j	number of the message
msg.a	byte	80	answer = DDLM_Upload
msg.f	byte	f	error, state
msg.b	byte	0	no command
msg.e	byte	0	extention, not used
msg[0...x-1]	byte string		data of whished data set file

The `msg.d[x]` area will contain the wished data field that was addressed within the command message. I

Possible values for `msg.f` are the following:

error number msg.f	signification
0	no error
19	CON_NO unknow command, DEVICE needs newer firmware
20	CON_LR, requested data set not available
21	CON_IV, parameter fault in request
52	CON_NI, area_code unknown
53	CON_EA, overstep of the buffer length
55	CON_IP, faulty parameter detected
57	CON_SI, sequence error
59	CON_DI, data incomplete or faulty

4.2.3.4.1 The Download and Coding of the Master Parameter

variable name	type	explanation
Bus_Para_Len	word	length of bus parameters inclusive the data length indicator: 32dec
FDL_Add	byte	own master address of the DEVICE
Baud_rate	byte	specifies the baud rate
Tsl	word	all following values can not be explained at this point. These are all PROFIBUS specific physical parameter values, changing the PROFIBUS behavior of the master as well of the slave basically. See PROFIBUS norm specification for further explanations.
min Tsdr	word	
max Tsdr	word	
Tqui	byte	
Tset	byte	
Ttr	long	
G	byte	
HSA	byte	
max_retry_limit	byte	
Bp_Flag	byte	switch to activate autoclear or not
Min_slave_intervall	word	this value guarantees the minimum time between two slave list cycles,
Poll_Timeout	word	maximum time within a master request of an other master have to be collected
Data_Control_Time	word	minimum time the global bus status field is actualized in the dual-port memory
Alarm_Max	byte	maximum of alarms per DPV1-Slave
Max_User_Global_ Control	byte	maximum number of Global_Control request, allowed to be started by the HOST at the same time
Octect[4]	byte array	reserved field

The FDL address specifies the address of this master station. It can be a value between 0-125. The addresses 126,127 are used for other special PROFIBUS components. It is commended to use the lowest possible address 1.
The baud rate operator has the following possible entries:

```
#define DP_BAUD_96        0   :9600kBaud
#define DP_BAUD_19_2      1   :19.2kBaud
#define DP_BAUD_93_75     2   :93.75kBaud
#define DP_BAUD_187_5     3   :187.5kBaud
#define DP_BAUD_500       4   :500kBaud
#define DP_BAUD_1500      6   :1.5MBaud
#define DP_BAUD_3000      7   :3.0MBaud
#define DP_BAUD_6000      8   :6.0MBaud
#define DP_BAUD_12000     9   :12MBaud
```

The parameter Bp_flag defines the system behavior, if the master recognizes that a slave station in not reponsive any more. If auto clear is activated the master shuts down the communication to all other slave modules too, else he keeps on the communication to the other station.

Bp_Flag

D7	D6	D5	D4	D3	D2	D1	D0
Err_Act							

reserved for further use

Error_Action_Flag: logical 1 -> change the operational state from OPERATE to STOP, if one station in not responsive any more

The minimum slave intervall value (in multiples of 100µs) must be collected from all slave specific values (defined in the GSD file of each slave) and the maximum value of all these defines the minimum slave intervall time. This time defines the time delay which the master has to wait until he can start the next DP cycle to all slaves. This mechanism suppresses data or interrupt overflow in the slave stations. So the slowest slave station decides on the whole PROFIBUS DP performance.

The data control time (in multiples of 10ms) defines the update rate of the global bus status field in the dual-port memory and the cycle time of the PROFIBUS-DP broadcast command global control, which has to be sent to report the internal master state to all stations.

The download of the master parameters

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	35	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	68	command = DDLM_Download
	msg.e	byte	k	extension
Service header	msg.d[0]	byte	0	Req_Adr, not used
	msg.d[1]	byte	127	Area_Code
	msg.d[2]	word	0	Add_Offset
DPM_BUS_DP	msg.d[4]	word	32	Bus_Para_Len
	msg.d[6]	byte	1-125	FDL_Add
	msg.d[7]	byte	0-9	Baud_rate
	msg.d[8]	word	37-16383	TSL
	msg.d[10]	word	1-1023	min TSDR
	msg.d[12]	word	1-1023	max TSDR
	msg.d[14]	byte	0-127	TQUI
	msg.d[15]	byte	1-255	TSET
	msg.d[16]	dword	255-...	TTR
	msg.d[20]	byte	1-255	G
	msg.d[21]	byte	1-126	HSA
	msg.d[22]	byte	0-7	max_retry_limit
	msg.d[23]	byte	0-255	Bp_Flag
	msg.d[24]	word	0-65535	Min_Slave_Intervall
	msg.d[26]	word	0-65535	Poll_Timeout
	msg.d[28]	word	1-65535	Data_Control_Time
	msg.d[30]	byte	32	Alarm_Max
	msg.d[31]	byte	1	Max_User_Global_Control
	msg.d[32...34]	byte array		reserved

See the below the corresponding structure in the header file:

DPM_BUS_DP

4.2.3.4.2 Download and Coding of the Slave Parameter Data Sets

variable name	type	explanation
Slave_Para_len	word	length of whole data set inclusive the length parameter
Sl_Flag	byte	in accordance to the DP extention to EN 50170 (DPV1 support)
Slave_Type	byte	always 0 for a DP-Slave
Max_Diag_ Data_len	byte	Maximum
Max_Alarm_Len	byte	Maximum length of a Alarm PDU
Max_Channel_ Data_Length	byte	maximum size of MSAC1-PDU
Diag_Upd_Delay	byte	number of DDLM_Slave_Diag while Prm_Req is still set
Alarm_Mode	byte	maximum number of possible active alarms
Add_Sl_Flag	byte	ignore auto_clear
Octet 1..6	octet string	reserved for further use
Prm_Data_Len	word	length of the following Prm. data inclusive the length of the size indicator
Prm_Data	octet string	see corresponding HEADER file for structure
Cfg_Data_Len	word	length of the following Cfg. data inclusive the length of the size indicator
Cfg_Data	octet string	see corresponding HEADER file for structure
Add_Tab_Len	word	length of the following add. tab data inclusive the length of the size indicator
Add_Tab	octet string	see corresponding HEADER file for structure
Slave_User_Data_ Len	word	length of the following slave user specific data inclusive the length of the size indicator
Slave_User_Data	octet string	see corresponding HEADER file for structure

This structure corresponds to PROFIBUS-norm EN 50170 chapter 'coding of the slave parameters' structure.
The `Prm_data` field includes the parameter data block which will be sent to the slave station while its startup procedure with the PROFIBUS-DP command 'Set_Prm'. This field includes 7 byte norm specific parameters and a `User_ prm_data` field as extended user specific data.
The `Cfg_data` field includes the configuration data of the slave, which decide on the number of input and outputs of the slave. This data is sent to the slave with the PROFIBUS-DP command 'Check_Cfg' to induce the slave to compare this configuration with its own internally saved one.
The `Add_tab` field is a Hilscher specific field which configures the different offset addresses of process data within the dual-port memory for modular and simple I/O slaves in common. See the following structure:

variable name	type	explanation
Input_Count	byte	number of input offsets following in the IO_Offset table
Output_count	byte	number of output offsets following in the IO_Offset table
IO_Offsets[...]	word array	word or byte IO_Offset in the order: first all input offsets then all output offsets in case of modular stations

One module entry in the `Cfg_list` must result a corresponding entry in the `Add_Tab`, except modules with the data length of 0. In this table the offset address within the dual-port memory of each module is held down. If the upper bit 15 in the `IO_Offset[...]` value is set to logical '1' then the address is interpreted by the DEVICE as byte offset address, else it is interpreted as word offset address.

Download example of a slave parameter data set with the address 4, without using the sequenced download procedure.

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	???	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	68	command = DDLM_Download
	msg.e	byte	k	extension
Service Header	msg.d[0]	byte	0	Req_Adr, not used
	msg.d[1]	byte	4	Area_Code, Slave address
	msg.d[2]	word	0	Add_Offset
Slave data set	msg.d[4]	word	???	Slave_Para_Len
	msg.d[6]	byte	128	Sl_Flag
	msg.d[7]	byte	0	Slave_Typ
	msg.d[8]	12 bytes	0	Octet1 - Octet12 described in EN50170
	msg.d[20]	word	???	Prm_Data_Len
	msg.d[...]	byte string	???	Prm_Data
	msg.d[...]	word	???	Cfg_Data_Len
	msg.d[...]	byte string	???	Cfg_Data
	msg.d[...]	word	???	Add_Tab_Len
	msg.d[...]	byte string	???	Add_Tab
	msg.d[...]	word	???	Slave_User_Data_len
	msg.d[...]	byte string	???	Slave_User_Data

The coding at `msg.d[4]` corresponds to PROFIBUS-norm EN 50170 in the chapter 'coding of the slave parameters' structure.
The structure of the slave parameters could not be laid down static, because the data in `msg.d[XX]` causes different lengths. Therefore no obvious addresses can be fixed to the several start addresses of the different parameters.

See below the corresponding structures in the header `DPM_USER.H` file:

DPM_SL_PRM_HEADER
DPM_SL_PRM_PRM_DATA
DPM_SL_PRM_CFG_DATA
DPM_SL_PRM_ADD_TAB
DPM_SL_PRM_USR_DATA

4.2.3.5 Example of Message Device Parameter Data Sets hexdecimal

Input and Output 4 Word module, Input and Output 10 Word module

03 10 2F 08 00 00 44 00, message header
00 02 00 00, rem adr = 2 etc.
2B 00 80 00 00 00 00 00 00 00 00 00 00 00 00 00, Header, Length = 0x002B
09 00 88 02 02 0B 1D 80 00, Parameter Data = Length 0x0009
04 00 F3 F9, Config. Data Length = 0x0004, 4 Word I/O, 10 Word I/O
0c 00 02 02 00 00 04 00 00 00 04 00, IO-Offset, offset Word 0 and Word 4
02 00, Empty slave user data

AEG TIO 16 Bit I/O

03 10 30 08 00 00 44 00, message header
00 05 00 00, rem adr = 5 etc.
2C 00 80 00 00 00 00 00 00 00 00 00 00 00 00 00, Header, Length = 0x002C
0E 00 88 02 02 0B 33 44 00 00 00 00 00 00, Parameter Data = Length 0x000E
04 00 21 11, Config. Data Length = 0x0004, 2 Byte Output, 2 Byte input
08 00 01 01 11 00 10 00, IO-Offset = O-offset Word 10 and I-offset Word 11
02 00, Empty slave user data

4.2.4 DDLM Online DP-Master - DP-Slave Functions

4.2.4.1 DDLM_Global_Control

	command message			
Message Header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	3	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	70	command = DDLM_Global_Control
	msg.e	byte	0	extension
DDLM_GLOBAL_CONTROL_ REQUEST	msg.d[0]	byte	0-127	Rem_Add
	msg.d[1]	byte	c	Control_Command
	msg.d[2]	byte	g	Group_Select

This command makes it possible, to send commands at one or several DP slaves. A DP slave accepts a control command only of the DP master which has parameterized the DP slave.

The parameter `Rem_Adr` fixes the address of the slave. The value could be from 0 to 127. The value of 127 is the special global address. If this address is selected all slaves are influenced simultaneously.

The parameter `Control_Command` fixes the command to be send.

Control_Command

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Sync	Unsync	Freeze	UnFreeze	Clear_Data	0

- D0: reserved
- D1: clear output data
- D2: unfreeze input data
- D3: freeze input data
- D4: neutralize the sync command
- D5: freeze output data, till sync commando is neutralized
- D6, D7: reserved

Combinations of the bits unsync/sync und unfreeze/freeze:

bit 2 or 4	bit 3 or 5	signification
0	0	no function
0	1	function will be activated
1	0	function will be inactive

1	1	function will be inactive

The parameter `Group_Select` decides, which group of the possible 8 ones is addressed. The command will be activated in the slave device, if the AND knotting between its internal `Group_Ident` (parameter which was configured by the master in the startup phase of the slave) and the wished `Group_Select` is a logical '1'. If `Group_Select` contains the value 0, no group selection (AND knotting) is done by the slave device when it receives the command.

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF-Task
	msg.ln	byte	1	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	70	answer = DDLM_Global_Control
	msg.f	byte	0	error, state
	msg.b	byte	0	no command
	msg.e	byte	0	extension
DDLM_GLOBAL_CONTROL_ CONFIRM	msg.d[0]	byte	0-127	Rem_Add

The `DDLM_Global_Control` command will be sent with a multicast command. Therfore this command always have a successing execute and no error will be placed in `msg.f` of the answer message.

See below the corresponding structure in the header file:

DDLM_GLOBAL_CONTROL_REQUEST
DDLM_GLOBAL_CONTROL_CONFIRM

4.2.4.2 DDLM_Slave_Diag

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	8	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	66	command = DDLM_Slave_Diag
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	k	Rem_Adr
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	0	data address, unused
	msg.data_idx	byte	0	data index, unused
	msg.data_cnt	byte	0	data count, unused
	msg.data_type	byte	10	data type byte string TASK_TDT_STRING
	msg.function	byte	1 3	function read from internal Buffer TASK_TFC_READ function read directly from slave TASK_TFC_QUERY

The command serves to read out the diagnostic structure of a slave. The wished address of the slave must be placed in `msg.device_adr` corresponding to the real address of the slave within the network.
The `msg.function` decides if either the diagnostic data is taken from the internal buffer without causing a network access to the slave, or if the service is sent to the slave directly and data of the response is sent back in the answer message. Calling the command without network access `msg.function = TASK_TFC_READ`, is much more faster in reaction time for getting the answer, then the other mode. But to use this mode makes only sense for slaves which are handled by the master. Only for that slaves the master drives the buffer update mechanism on every received new diagnostic data. The corresponding bit of each master assigned slave within the `Sl_Diag` field in the global bus status field indicates, if a changed diagnostic information since the last request is available and should be requested and read out. The diag bit of a slave station will be cleared on each `DDLM_Slave_Diag.Req` that is performed. `Sl_Diag`-bits of slaves which are not assigned to the master are not influenced by this mechanism in general.
The `Sl_Diag` field will be updated in relation to the executed main PRFOFIBUS master cycle time, that is needed to update all input and output data. If a slave station then is permanent not available in the network for example, the corresponding diag bit within this field could be set every 500µsec again and again to indicate that the slave is not present.

	answer message			
	variable	type	value	signification
Message header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF-Task
	msg.ln	byte	8+x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	66	answer = DDLM_Slave_Diag
	msg.f	byte	0	error, state
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	k	Rem_Adr
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	0	data address, unused
	msg.data_idx	byte	0	data index, unused
	msg.data_cnt	byte	x	data count = length of diagnostic structure
	msg.data_type	byte	5	data type bytestring TASK_TDT_UINT8
	msg.function	byte	1	function read TASK_TFC_READ
DPM_SLAVE_SINGLE_ DIAGNOSTICS	msg.d[0]	byte	S1	Stationsstatus_1
	msg.d[1]	byte	S2	Stationsstatus_2
	msg.d[2]	byte	S3	Stationsstatus_3
	msg.d[3]	byte	MA	master address
	msg.d[4-5]	word	ID	ident number
	msg.d[6...(x-1)]	byte string	EX	extended diagnostic

The confirmation message informs the HOST application about success or fail of the `DDLM_Slave_Diag` service.

msg.f	signification	error source	help
0 = CON_OK	service could be executed without an error		
17 = CON_NA	no response of the station	DEVICE	check network wiring, check bus address of slave or baud rate support
18 = CON_DS	master not into the logical token ring	network in general	check master DP-Address or highest-station-Address of other masters. Examine bus wiring to bus short circuits.
161 = TASK_F_ DEVICE_ADR	Remote Addres in request service out of range	HOST	Check parameter in request message

REMARK: The internal buffer of the DEVICE can handle 6 Bytes standard and 100 Bytes Extended Diagnostic data per slave device.

Octet 1: Stationsstatus_1

D7	D6	D5	D4	D3	D2	D1	D0
Master_ Lock	Prm_ Fault	Inv._Sl_ Res.	Not_ Supp.	Ext_ Diag	Cfg_ Fault	Sta._No t_Rdy	Sta._No n_Exist.

- D0: slave not responding
- D1: slave not ready
- D2: slave is wrong paramterized
- D3: the area Ext_Diag is used for extended diagnostic
- D4: unknown command detected by the slave
- D5: inplausible response of the slave
- D6: last parameter telegram faulty
- D7: slave is parameterized by another master

Octet 2: Stationsstatus_2

D7	D6	D5	D4	D3	D2	D1	D0
Deakti- vated	res- served	Sync_ Mode	Freeze_ Mode	WD_On	1	Stat_ Diag	Prm_ Req

- D0: slave must be parameterized
- D1: get diagnostic from slave, till bit is released
- D3: watchdog activated
- D4: freeze-command active
- D5: sync-command active
- D6: reserved
- D7: slave not projected

Octet 3: Stationsstatus_3

D7	D6	D5	D4	D3	D2	D1	D0
Ext_Diag _Ovfl.							

- D6: reserved
- D7: the slave has more diagnostic data available than it can send

Octet 4: Master_Add

This byte contains the master address from the master which has done the parameterization of the slave. If a slave is not parameterized the value is 255.

Octet 5_6: Ident_Number

In this byte the slave station reports its ident number.

Octet 7_32: Ext_Diag_Data

This is an extented diagnostic buffer. The values are fixed in the manual of the slave station or can be read in the PROFIBUS norm specification.

See below the corresponding structure in the header file:

DPM_SLAVE_SINGLE_DIAGNOSTICS

4.2.4.3 DDLM_Set_Prm

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	1 + x	length of message
	msg.nr	byte	0	number of the message, unused
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	74	command = DDLM_Set_Prm
	msg.e	byte	0	extension
DDLM_SET_PRM_REQUEST	msg.d[0]	byte	k	Rem_Adr
	msg.d[1... 234]	byte string		Usr_Prm_Data

This service activates the `DDLM_Set_Prm` DP-Norm primitive to send parameter data to a specific slave component during runtime.
The DEVICE builds up the parameter that are sent to the slave by adding the `Usr_Prm_Data` field coming from the message to the 7 bytes (ident_number, watchdog_Factor, group_Ident etc.) standard slave parameter coming from the internal configuration. Therfore the service can only be proceeded by the DEVICE if the corresponding slave component is already configured either via SyCon configuration tool or the `DDLM_Download` message procedure to have this 7 byte standard header parameter data available. The HOST has no influence on this 7 bytes parameter any more during runtime.

The specific DP-Slave address the service shall be addressed to must be fixed in the `Rem_Adr` location. The additional HOST specific parameter must be inserted in the `Usr_Prm_Data` field.

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF-Task
	msg.ln	byte	1	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	74	answer = DDLM_Set_Prm
	msg.f	byte		error, state, see table below
	msg.b	byte	0	no command
	msg.e	byte	0	extension
DDLM_SET_PRM_CONFIRM	msg.d[0]	byte	0-127	Rem_Add

The confirmation message informs the HOST application about success or fail of the `DDLM_Set_Prm` service.

msg.f	signification	error source	help
0 = CON_OK	service could be executed without an error		
2 = CON_RR	resource unavailable	slave	slave has no left buffer space for the requested service
3 = CON_RS	requested function of master is not activated within the slave	slave	Remote SAP is not activated
17 = CON_NA	no response of the station	slave	check network wiring, check bus address of slave or baud rate support
18 = CON_DS	master not into the logical token ring	network in general	check master DP-Address or highest-station-Address of other masters. Examine bus wiring to bus short circuits.
54 = CON_AD	negative response received, acces denied	slave	access denied to requested data. Check Error_Code_1 and Error_Code_2 in reponse message to get closer error information

4.2.4.4 DDLM_RD_Input

With this function the HOST can read the input process data of a connected slave independant if the slave is assigned and communicating to an other master or not.

command message			
variable	type	value	signification
msg.rx	byte	7	receiver = FDL-Task
msg.tx	byte	16	transmitter = user at HOST
msg.ln	byte	8	length of message
msg.nr	byte	j	number of the message
msg.a	byte	0	no answer
msg.f	byte	0	error, state
msg.b	byte	0xE9	command = DDLM_RD_Input
msg.e	byte	0	extension
msg.DeviceAdr	byte	k	Rem_Adr
msg.DataArea	byte	0	data area, unused
msg.DataAdr	word	0	data address, unused
msg.DataIdx	byte	0	data index, unused
msg.DataCnt	byte	x	data count = length of Inputdate of module. The value will be overwritten by the true value.
msg.DataType	byte	5	data type byte string = TASK_TDT_UINT8
msg.Function	byte	1	function read = TASK_TFC_READ

answer message			
variable	type	value	signification
msg.rx	byte	16	receiver = user at HOST
msg.tx	byte	7	transmitter = FDL-Task
msg.ln	byte	8+x	length of message
msg.nr	byte	j	number of the message
msg.a	byte	0xE9	answer = DDLM_RD_Input
msg.f	byte	0	error, state
msg.b	byte	0	no command
msg.e	byte	0	extension
msg.DeviceAdr	byte	k	Rem_Adr
msg.DataArea	byte	0	data area, unused
msg.DataAdr	word	0	data address, unused
msg.DataIdx	byte	0	data index ,unused
msg.DataCnt	byte	x	data count = length of input data, will be set by the read function, independent of the value in the request.
msg.DataType	byte	5	data type bytestring = TASK_TDT_UINT8
msg.Function	byte	1	function read =TASK_TFC_READ
msg.d[0-243max]	byte		input data of the module

4.2.4.5 DDLM_RD_Output

With this function the HOST can read the output process data of a connected slave independant if the slave is assigned and communicating to an other master or not.

command message			
variable	type	value	signification
msg.rx	byte	7	receiver = FDL-Task
msg.tx	byte	16	transmitter = user at HOST
msg.ln	byte	8	length of message
msg.nr	byte	j	number of the message
msg.a	byte	0	no answer
msg.f	byte	0	error, state
msg.b	byte	0xEA	command = DDLM_RD_Output
msg.e	byte	0	extension
msg.DeviceAdr	byte	k	Rem_Adr
msg.DataArea	byte	0	data area, unused
msg.DataAdr	word	0	data address, unused
msg.DataIdx	byte	0	data index, unused
msg.DataCnt	byte	x	data count = length of output data of module The value will be overwritten by the true value.
msg.DataType	byte	5	data type byte string = TASK_TDT_UINT8
msg.Function	byte	1	function read = TASK_TFC_READ

answer message			
variable	type	value	signification
msg.rx	byte	16	receiver = user at HOST
msg.tx	byte	7	transmitter = FDL-Task
msg.ln	byte	8+x	length of message
msg.nr	byte	j	number of the message
msg.a	byte	0xEA	answer = DDLM_RD_Output
msg.f	byte	0	error, state
msg.b	byte	0	no command
msg.e	byte	0	extension
msg.DeviceAdr	byte	k	Rem_Adr
msg.DataArea	byte	0	data area, unused
msg.DataAdr	word	0	data address, unused
msg.DataIdx	byte	0	data index, unused
msg.DataCnt	byte	x	data count = length of output data, will be set by the read function, independent of the value in the request.
msg.DataType	byte	5	data type byte string TASK_TDT_UINT8
msg.Function	byte	1	function read TASK_TFC_READ
msg.d[0-243max]	byte		output data of the module

4.2.5 DPV1 Class 1 Services

The new acyclic functionality of DPV1 requires a flexible and enhanced addressing scheme of data within field devices.
DPV1 devices are composed of so called SLOTS regarding the acyclic accessible data. These slots do not have to be physical objects. Futhermore, they may represent physical blocks of data. Slots contain information which is index addressable.
Index addressable data represents variables or compact data blocks. Index data is accessed by means of the Read and Write services and tramitted in a single frame.
The concrete use of `Slot_Number` and `Index` to address data blocks within the device is manufacturer-specific.

In DPV1 it is possible to transfer alarm and status messages in addition to diagnostic data as described in EN 50170. The alarm model allows the transfer of an alarm from the DP-Slave to the DP-Master (Class 1) and the explicit acknowlegdement of the alarm by the master by means of the confirmed HOST service `MSAL1_Alarm_Con`. The alarm model is slot (typically module) based. For the tranmission of an alarm following conditions have to be fulfilled:

- the slave has to be in the DATA-EXCHANGE mode
- the MSAC_C1 connection has to be open (`DPV1_Enable` = TRUE)
- the corresponding Alarm_Type has to be enabled (by means of the bits `Enable_xxx_Alarm` in the service `DDLM_Set_Prm`)
- the limit of active alarms in not exceeded

Two options of parallel alarm handling are defined:

- only one alarm of a specific Alarm_Type can be active at one time (fixed in the `DDLM_Set_Prm` service)
- Several alarms (2 to 32) of the same or different type can be active at one time (fixed in `DDLM_Set_Prm` service)

An alarm will be queued and therefore a previous alarm will never be overwritten. The alarm remains active until the master respectivly the HOST explicitly acknowledges the alarm.

Note: The DPV1 functions described below are only available within the following DEVICEs: EC1 based Products , CIF30-PB, CIF 104-PB, COM-PB, CIF 50-PB, CIF 60-PB with firmware version >= V1.040
The services are not implemented in firmware version of the following DEVICEs in general: CIF 30-DPM, CIF 104-DPM/DPMR, COM-DPM

4.2.5.1 MSAC1_Read_Req

By means of this service a read request for a data block is transferred to a DPV1-Slave. The services operates slot and index related.

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	3	receiver = USR_INTF
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	8	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0x11	command = MSAC1_Read_Write
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0...126	Rem_Add
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0...254	Slot_Number
	msg.data_idx	byte	0...254	Index
	msg.data_cnt	byte	x 0...240	length of data block to be read
	msg.data_type	byte	10	data type, byte string = TASK_TDT_STRING
	msg.function	byte	1	function MSAC1_Read = TASK_TFC_READ

The parameter `msg.device_adr = Rem_Add` contains the station address of the slave responder.
The parameter `msg.data_adr = Slot_Number` is used in the destination device for addressing the desired data block slot (typically a module).
The parameter `msg.data_idx = Index` is used in the destination device for addressing the desired data block.
The parameter `msg.data_cnt = Length` indicates the number of bytes of the data block that has to be read. If the server data block length is less than requested then the length in the response will be the actual length of the data block. If the server data block length is greater or equal then requested then the reponse contains the same length of data. The DP-slave may answer with an error response if the data access is not allowed.

4.2.5.2 MSAC1_Read_Con

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF
	msg.ln	byte	8 10 8+x	length of message if msg.f <> 0 and msg.f <> CON_AD if msg.f = CON_AD if msg.f = 0
	msg.nr	byte	j	number of the message
	msg.a	byte	0x11	answer = MSAC1_Read_Write
	msg.f	byte	z	error, state, see table below
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0...126	Rem_Add
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0...254	Slot_Number
	msg.data_idx	byte	0...254	Index
	msg.data_cnt	byte	x 0...240	length of data block, will be set to the real value of received data
	msg.data_type	byte	10	data type, byte string = TASK_TDT_STRING
	msg.function	byte	1	function MSAC1_Read = TASK_TFC_READ
	If 'msf.f' = CON_AD			
	msg.d[0]	byte		Error_Code_1
	msg.d[1]	byte		Error_Code_2
	If 'msf.f' = 0			
	msg.d[0]	byte		1'st data byte from slave
	...	...		...
	msg.d[x-1]	byte		last data from slave

4.2.5.3 MSAC1_Write_Req

By means of this service a write request is transferred to a DPV1-Slave

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	3	receiver = USR_INTF
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	8+x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0x11	command = MSAC1_Read_Write
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0...126	Rem_Add
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0...254	Slot_Number
	msg.data_idx	byte	0...254	Index
	msg.data_cnt	byte	x 0...240	length of data block to be written
	msg.data_type	byte	10	data type, byte string = TASK_TDT_STRING
	msg.function	byte	2	function MSAC1_Write = TASK_TFC_WRITE
WRITE_RAW_DATA	msg.d[0]	byte		1'st data byte to slave
	...	...		...
	msg.d[x-1]	byte		last data byte to slave

The parameter `msg.device_adr = Rem_Add` contains the station address of the slave responder.
The parameter `msg.data_adr = Slot_Number` is used in the destination device for addressing the desired data block slot (typically a module).
The parameter `msg.data_idx = Index` is used in the destination device for addressing the desired data block.
The parameter `msg.data_cnt = Length` indicates the number of bytes of the data block that has to be written. If the destination data block length is less than required then the reply will conatin an error message. If the data block length is greater or equal than the required length the reply conatins the number of bytes which have been written. The DP-Slave may answer with an error response if the data access is not allowed.
The field `msg.d[x]` contains the data block which has to be written and consists of the number of octets indicated in the length of the request.

4.2.5.4 MSAC1_Write_Con

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF
	msg.ln	byte	8 10	length of message, if msg.f = 0 if msg.f = CON_AD
	msg.nr	byte	j	number of the message
	msg.a	byte	0x11	answer = MSAC1_Read_Write
	msg.f	byte	z	error, state, see table below
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0...126	Rem_Add
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0...254	Slot_Number
	msg.data_idx	byte	0...254	Index
	msg.data_cnt	byte	x 0...240	length of data block that was written
	msg.data_type	byte	10	data type, byte string = TASK_TDT_STRING
	msg.function	byte	2	function MSAC1_Write = TASK_TFC_WRITE
ERROR_CODE	If 'msf.f' = CON_AD			
	msg.d[0]	byte		Error_Code_1
	msg.d[1]	byte		Error_Code_2

4.2.5.4.1 Error Response Definitions

The error coding scheme of DPV1 uses error codes for classification of the error and indication of an additional error code. Basic communication errors which are not related to the DPV1 specification are reported in `msg.f` of each answer message:

msg.f	signification	error source	help
0 = CON_OK	service could be executed without an error		
2 = CON_RR	resource unavailable	slave	slave has no left buffer space for the requested service
3 = CON_RS	requested function of master is not activated within the slave	slave	slave is not activated in its DPV1support
9 = CON_NR	no answer-data, although the slave has to reponse with data	slave	slave hasn't sent back any amount of data
17 = CON_NA	no response of the station	slave	check network wiring, check bus address of slave or baud rate support
18 = CON_DS	master not into the logical token ring	network in general	check master DP-Address or highest-station-Address of other masters. Examine bus wiring to bus short circuits.
25 = CON_NP	no plausible reaction of remote partner	slave	slave does not work DPV1 norm conform
54 = CON_AD	negative response received, acces denied	slave	access denied to requested data. Check Error_Code_1 and Error_Code_2 in reponse message to get closer error information
0x81 = REJ_SE	DEVICE is about to stop the DPV1-communication or the DPV1 is not in OPEN state	HOST, configuration	DPV1 communications must be configured to be activated by the DEVICE
0x82 = REJ_ABORT	DEVICE has stopped the DPV1-communication automatically	slave	a previous addressed slave has responded with non conform parameters
0x83 = REJ_PS	a previous service is still in process	HOST	wait for the outstanding answer first. Parallel services are not allowed
0x84 = REJ_LE	the length indicator msg.data_cnt oversteps maximum configured size	HOST	reduce length of message or enlarge maximum buffer size in SyCon or in slave data set

0x85 = REJ_IV	wrong parameter in request	HOST	check msg.function or msg.device_adr parameter of requested message
0x9a = REJ_COM	unknown msg.b command	HOST	correct the reqested msg.b parameter of message

In case of error the two `Error_Codes` represent further detailed error information:

Error_Code_1:

D7	D6	D5	D4	D3	D2	D1	D0
Error_Class				Error_Code			

Error_Class	Meaning	Error_Code
0 to 9 =	reserved	
10 =	application	0 = read error 1= write error 2 = module failure 3 to 7 = reserved 8 = version conflict 9 = feature nor supported 10 to 15 = user specific
11 =	access	0 = invalid index 1 = write length error 2 = invalid slot 3 = type conflict 4 = invalid area 5 = state conflict 6 = access denied 7 = invalid range 8 = invalid parameter 9 = invalid type 10 to 15 = user specific
12 =	resource	0 = read constrain conflict 1 = write constrain conflict 2 = resource busy 3 = resource unavailable 4 to 7 = reserved 8 to 15 = user specific
13 to 15 =	user specific	

Error_Code_2 is fully user specific and can not be defined here.

4.2.5.5 MSAL1M_Alarm_Ind

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF
	msg.ln	byte	8+x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0x12	command = MSAL1M_Alarm_Ind
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0...126	Rem_Add
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0...254	Slot_Number
	msg.data_idx	byte	0...31	Seq_Nr
	msg.data_cnt	byte	0...59	length of Diagnostic_User_Data
	msg.data_type	byte	1-6, 32-126	Alarm_Type
	msg.function	byte	0...7	Alarm_Spec_Ack
DIAGNOSTIC_USER_DATA	msg.d[0]	byte		first additional manufacturer specific alarm info
	...	...	...	...
	msg.d[x-1]	byte		last additional manufacturer specific alarm info

By means of this service and alarm message is transferred from the DPV1-Slave to the DPV1-Master.
The `msg.d[0]` field will contain the received diagnostic informations transparently.
`msg.device_adr` parameter contains the station address which indicates the alarm.
`msg.data_type` identifies the alarm type. Coding see in the table below.
`msg.data_adr` = Slot_Number indicates the source of the alarm. The range here goes from 0 to 254. The value 255 is declared as reserved in the DPV1 norm description.
By means of the `msg.data_idx` = Seq_Nr an unique identification of an alarm message is accomplished. Valid range is 0 to 31.
The `msg.function` = Alarm_Spec_Ack parameter gives additional alarm information, e.g. an alarm apears, disappears or no further differentiation is not possible or if the alarm requires an additonal user acknowledge. See table below for further explanations.

Alarm_Spec_Ack

D7	D6	D5	D4	D3	D2	D1	D0
					Add_Ack	Alarm_Specifier	

Alarm_Specifier:
00 no further differentiation
01 Error appears and Slot disturbed
10 Error disappears and Slot is okay
11 Error disappears and Slot is still disturbed

Add_Ack: this alarm requires in addtional to the MSAL1M_ALARM_RES a separate user acknowledge. This can be done for instance by means of a write service.

reserved

Alarm_Type

Alarm_Type value	explanation
0	reserved
1	Diagnostic_Alarm
2	Process_Alarm
3	Pull_Alarm
4	Plug_Alarm
5	Status_Alarm
6	Update_Alarm
7 - 31	reserved
32 - 126	manufactuerer-specific
127	reserved

4.2.5.6 MSAL1M_Alarm_Res

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	3	receiver = USR_INTF
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	8	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0x12	command = MSAL1M_Alarm_Res
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0...126	Rem_Add
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0...254	Slot_Number
	msg.data_idx	byte	0...31	Seq_Nr
	msg.data_cnt	byte	0	data_cnt, unused
	msg.data_type	byte	1-6, 32-126	Alarm_Type
	msg.function	byte	0...7	Alarm_Spec, Add_ack

With this command the HOST has to acknowledge a previous received `MSAL1M_Alarm_Ind` message indication. The DEVICE will then send the corresponding `MSAC1M_Alarm_Ack` PROFIBUS-DP message to the slave device to inform it that the HOST application has received and acknowledged the alarm.

4.2.5.7 MSAL1M_Alarm_Con

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	3	receiver = USR_INTF
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	8	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0x12	answer = MSAL1M_Alarm_Con
	msg.f	byte	0	error, state
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0...126	Rem_Add
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0...254	Slot_Number
	msg.data_idx	byte	0...31	Seq_Nr
	msg.data_cnt	byte	0	data_cnt, unused
	msg.data_type	byte	1-6, 32-126	Alarm_Type
	msg.function	byte	0...7	Alarm_Spec, Add_ack

With this answer message the HOST gets the confirmation for its previous send `MSAL1M_Alarm_Res`. If no error is indicated the master has successfully sent back the `MSAL1M_Alarm_Ack` PROFIBUS message to the slave device.

msg.f	signification	error source	help
0x86 = REJ_INT	the alarm handler is not initialized	DEVICE	no DPV1 capable device configured within the card
0x87 = REJ_SRT	the alarm handler is currently stopped	DEVICE	no DPV1 capable slave device is in process data exchange with the DEVICE. Check if network is running
0x88 = REJ_ENA	the alarm that shall be acknowledged is not enabled in slave parameter data	HOST	enable the corresponding alarm in slave configurtion data set
0x89 = REJ_NPD	the alarm that shall be acknowledge is not pending on a MSAL1_Alarm_Res	HOST	check the parameter Alarm_Type and Seq_Nr. Both must be equal to the MSAL1_Alarm_ind parameter
0x9a = REJ_COM	unknown msg.b command	HOST	correct the reqested msg.b parameter of message

4.2.6 Class 2 Services

Additional communication relationships of DP-Slave to DP-Masters can be done via socalled Class 2 services.
The DPV1 master (Class 2) uses the service `MSAC2M_Initiate` to establish and the `MSAC2M_Abor`t to abort a communication relationship to the DPV1-Slave.
By means of the service `MSAC2M_Read` a read request for a data block is transferred to a DPV1-Slave. By means of the service `MSAC2M_Write` a write request is transferred to a DPV1-Slave.

4.2.6.1 DDLM_Life_List

This command primitive requests an up-to-date list of all stations that may be currently reached on the bus.

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	7	receiver = FDL-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	8	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	0x96	command = DDLM_Life_List
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0	device_adr, unsued
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0	data_adr, unused
	msg.data_idx	byte	0	data_idx, unused
	msg.data_cnt	byte	0	data_cnt, unused
	msg.data_type	byte	10	data type, byte string TASK_TDT_STRING
	msg.function	byte	1	function code, read TASK_TFC_READ

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	7	transmitter = FDL-Task
	msg.ln	byte	135	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0x96	command = DDLM_Life_List
	msg.f	byte	0	error, see table below
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0	device_adr, unsued
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0	data_adr, unused
	msg.data_idx	byte	0	data_idx, unused
	msg.data_cnt	byte	127	data_cnt, unused
	msg.data_type	byte	10	data type, byte string TASK_TDT_STRING
	msg.function	byte	2	function code, read TASK_TFC_READ

	msg.d[0-126]	array of bytes	0xff 0x30 0x00	status of each station: not available active station passive station

Error response defintions in `msg.f`:

msg.f	signification	error source	help
0 = CON_OK	service could be executed without an error		
18 = CON_DS	master not into the logical token ring	network in general	check master DP-Address or highest-station-Address of other masters. Examine bus wiring to bus short circuits.

4.2.6.2 DDLM_Set_Slave_Add

With this function it is possible to set the bus address of a slave using a message from the master. This function is not supported by all slaves.

command message			
variable	type	value	signification
msg.rx	byte	7	receiver = FDL-Task
msg.tx	byte	16	transmitter = user at HOST
msg.ln	byte	8 + 4 + x	length of message
msg.nr	byte	j	number of the message
msg.a	byte	0	no answer
msg.f	byte	0	error, state
msg.b	byte	0xEB	command = DDLM_Set_Slave_Add
msg.e	byte	0	extension
msg.DeviceAdr	byte	k	Rem_Adr
msg.DataArea	byte	0	data area, unused
msg.DataAdr	word	0	data address, unused
msg.DataIdx	byte	0	data index, unused
msg.DataCnt	byte	4 + x	data count = length of Input data of module The value will be overwritten by the true value.
msg.DataType	byte	5	data type byte string = TASK_TDT_UINT8
msg.Function	byte	2	function write = TASK_TFC_WRITE
msg.d[0]	byte		New_Slave_Address
msg.d[1]	byte		Ident_Number as an 'unsigned short'
msg.d[2]	byte		
msg.d[3]	byte	0; != 0	No_Add_Chg
msg.d[4 ... x]	byte		Rem_Slave_Data User specific data to store inside the slave. Values are dependent on the slave.

answer message			
variable	type	value	signification
msg.rx	byte	>=16	receiver = user at HOST
msg.tx	byte	7	transmitter = FDL-Task
msg.ln	byte	8	length of message
msg.nr	byte	j	number of the message
msg.a	byte	0xEB	answer = DDLM_Set_Slave_Add
msg.f	byte	see table	error, state
msg.b	byte	0	no command
msg.e	byte	0	extension
msg.DeviceAdr	byte	k	Rem_Adr
msg.DataArea	byte	0	data area, unused
msg.DataAdr	word	0	data address unused
msg.DataIdx	byte	0	data index unused
msg.DataCnt	byte	0	data count

answer message			
variable	type	value	signification
msg.DataType	byte	5	data type bytestring TASK_TDT_UINT8
msg.Function	byte	1	function read TASK_TFC_READ

Error response defintions in `msg.f`:

msg.f	signification	error source	help
0 = CON_OK	service could be executed without an error		
3 = CON_RS	requested function of master is not activated within the slave	slave	service not activated in slave
9 = CON_NR	no answer-data, although the slave has to reponse with data	slave	slave hasn't sent back any amount of data
17 = CON_NA	no response of the station	slave	check network wiring, check bus address of slave or baud rate support
18 = CON_DS	master not into the logical token ring	network in general	check master DP-Address or highest-station-Address of other masters. Examine bus wiring to bus short circuits.

4.2.6.3 DDLM_Get_Cfg

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	7	receiver = -FDL-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	8	length of message
	msg.nr	byte	0	number of the message, unused
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	232	command = DDLM_Get_Cfg
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	k	Rem_Adr
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	0	data address, unused
	msg.data_idx	byte	0	data index, unused
	msg.data_cnt	byte	0	data count , unused
	msg.data_type	byte	10	data type bytestring TASK_TDT_STRING
	msg.function	byte	1	function read TASK_TFC_READ

This service activates the `DDLM_Get_Cfg` DP-Norm primitive to get the current configuration data of a specific slave device.
The slave device that shall be addressed must be fixed in `msg.d[0]` byte.

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	7	transmitter = FDL-Task
	msg.ln	byte	8 + x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	232	answer = DDLM_Get_Cfg
	msg.f	byte		error, state, see table below
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	k	Rem_Adr
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	0	data address, unused
	msg.data_idx	byte	0	data index, unused
	msg.data_cnt	byte	x	data count = length of Real_Cfg_Data
	msg.data_type	byte	10	data type bytestring TASK_TDT_STRING
	msg.function	byte	1	function read TASK_TFC_READ
REAL_CFG_DATA	msg.[0...x]	byte array		Real_Cfg_data

The response message will contain the socalled real configuration data at message position msg.d[1...].

Error response defintions in `msg.f`:

msg.f	signification	error source	help
0 = CON_OK	service could be executed without an error		
3 = CON_RS	requested function of master is not activated within the slave	slave	service not activated in slave
9 = CON_NR	no answer-data, although the slave has to reponse with data	slave	slave hasn't sent back any amount of data
17 = CON_NA	no response of the station	slave	check network wiring, check bus address of slave or baud rate support
18 = CON_DS	master not into the logical token ring	network in general	check master DP-Address or highest-station-Address of other masters. Examine bus wiring to bus short circuits.

REMARK: The `REAL_CFG_DATA` can have a maximum length of 244 Bytes.

4.2.6.4 MSAC2M_Initiate_req

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	3	receiver = USR_INTF
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	1...x	length of message
	msg.nr	byte	CR	communication reference
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0x20	command = MSAC2M_Initiate
	msg.e	byte	0	extension
	msg.d[0]	byte	0...126	Rem_Adr
DPM_MSAC2M_INITIATE_REQ	msg.d[1...(x-1)]	structure		defined in DPV1-description on page 49

With this initiate service a Class2 connection to a specific slave must be established first before the services `MSAC2M_Read_req` or `MSAC2M_Write_req` can be used.
The parameter definition for this service beginning at position `msg.d[0]` within the message can be taken from the corresponding header file. The explanation of all available structure parameter can be read up in the PROFIBUS EN50170 (DPV1) norm description.
The initiate service sends the parameter directly to the slave who checks this parameter immediatly. If a slave does not support a wished service or does not support a defined profile, it will deny the initiate primitive. The connection state will go into the CLOSED state then. If the initiate request is confirmed positiv then the connection state go to OPEN and the master will now request IDLE-telegrams to the slave cyclically to supervise it, until the connection is aborted or until the connection is interrupted physically. In the last case the master will automatically abort and close the connection.
In a Class2 relationship can only exist one open connection at the same time, parallel established connections are forbidden. The DEVICE will deny further initiate request as long as a Class2 connection to a slave is in OPEN state. To initialize a connection to another slave device this already established connection must be aborted before with the `MSAC2M_Abort_req` primitive.
A connection is in OPENed state as long as the DEVICE has not send an `MSAC2M_Close_ind` message to the HOST. That means between this `MSAC2M_Initiate` message and the `MSAC2M_Close_ind` message all messages like `MSAC2M_Abort_ind` or `MSAC2_XXX_con_neg` messages must be collected by the HOST program too. Only when the `MSAC2M_Close_ind` messages was received, no further messages must be awaited by the HOST for this previously established Class2 connection.

4.2.6.5 MSAC2M_Initiate_con_pos

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF
	msg.ln	byte	1...x	length of message
	msg.nr	byte	CR	communication reference
	msg.a	byte	0x20	answer = MSAC2M_Initiate_con
	msg.f	byte	0	con_pos
	msg.b	byte	0	no command
	msg.e	byte	0	extension
	msg.d[0]	byte	0...126	Rem_Adr
DPM_MSAC2M_INITIATE_CON	msg.d[1...(x-1)]	structure		defined in DPV1-description

The positiv intiate response message has the value `msg.f` set to 0. The Class2 connection is now in established state. Because the connection is superviced from this position now on and a connection `MSAC2M_Abort` or a connection `MSAC2M_Close` can happen at any time, the HOST application has to look now permanently or at least cyclically if the `HostMailbox` of the DEVICE indicates an `MSAC2M_Abort_ind` or `MSAC2M_Close_ind` indication message or both messages one after the other. Because if a connection is ABORTed automatically through external influences this will be first indicated by an `MSAC2M_Abort_ind` message and because the connection is CLOSED then afterwards automatically too, the `MSAC2M_Close_ind` message indication will follow.
The structure `DPM_MSAC2M_INITIATE_CON` will be added to the message and comes from the positive response of the slave itself. This structure is described in the PROFIBUS specification and contains for example its supported profil or supported features.

4.2.6.6 MSAC2M_Initiate_con_neg

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF
	msg.ln	byte	3	length of message
	msg.nr	byte	CR	communication reference
	msg.a	byte	0x20	answer = MSAC2M_Initiate_con
	msg.f	byte	255	con_neg
	msg.b	byte	0	no command
	msg.e	byte	0	extension
	msg.d[0]	byte	0...126	Rem_Adr
ERROR_CODE	msg.d[1]	byte		Error_Code_1
	msg.d[2]	byte		Error_Code_2

In case of a negative initiate response of the slave the answer message contains the value 255 in the variable `msg.f`. Detailed error information can be found in `msg.d[1]` and `msg.d[2]`. The explanation of the different errors can be found in the Chapter 'Error Response Definitions'.

4.2.6.7 MSAC2M_Read_Req

By means of this service a read request via a Class2 connection for a data block is transferred to a slave. The services operates slot and index related.

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	3	receiver = USR_INTF
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	8	length of message
	msg.nr	byte	CR	communication reference
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0x21	command = MSAC2M_Read_Write_Data
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0...126	Rem_Add
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0...254	Slot_Number
	msg.data_idx	byte	0...254	Index
	msg.data_cnt	byte	x 0...240	length of data block to be read
	msg.data_type	byte	10	data type, byte string = TASK_TDT_STRING
	msg.function	byte	1	function MSAC2M_Read = TASK_TFC_READ

The parameter `msg.device_adr = Rem_Add` must contains the station address of the slave responder.
The parameter `msg.data_adr = Slot_Number` is used in the destination device for addressing the desired data block slot (typically a module).
The parameter `msg.data_idx = Index` is used in the destination device for addressing the desired data block.
The parameter `msg.data_cnt = Length` indicates the number of bytes of the data block that has to be read. If the server data block length is less than requested then the length in the response will be the actual length of the data block. If the server data block length is greater or equal then requested then the reponse contains the same length of data. The DP-slave may answer with an error response if the data access is not allowed.

4.2.6.8 MSAC2M_Read_con_pos

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF
	msg.ln	byte	8+x	length of message
	msg.nr	byte	CR	communication reference
	msg.a	byte	0x21	answer = MSAC2M_Read_Write_Data
	msg.f	byte	0	CON_POS
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0...126	Rem_Add
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0...254	Slot_Number
	msg.data_idx	byte	0...254	Index
	msg.data_cnt	byte	x 0...240	length of data block, will be set to the real number of received data bytes
	msg.data_type	byte	10	data type, byte string = TASK_TDT_STRING
	msg.function	byte	1	function MSAC2M_Read = TASK_TFC_READ
READ_RAW_DATA	msg.d[0]	byte		1'st data byte from slave
	...	...		...
	msg.d[x-1]	byte		last data from slave

If the slave responses with valid data then the `msg.f` of the DEVICEs answer message is equal 0. The real number of received data bytes is copied into `msg.data_cnt`. The received raw data itself is copied transparently into the `msg.d[...]` area.

4.2.6.9 MSAC2M_Read_con_neg

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF
	msg.ln	byte	8+2	length of message
	msg.nr	byte	CR	communication reference
	msg.a	byte	0x21	answer = MSAC2M_Read_Write_Data
	msg.f	byte	255	CON_NEG
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0...126	Rem_Add
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0...254	Slot_Number
	msg.data_idx	byte	0...254	Index
	msg.data_cnt	byte	2	two error bytes
	msg.data_type	byte	10	data type, byte string = TASK_TDT_STRING
	msg.function	byte	1	function MSAC2M_Read = TASK_TFC_READ
ERROR_CODE	msg.d[0]	byte		Error_Code_1
	msg.d[1]	byte		Error_Code_2

If the slave denies the read access the variable `msg.f` is set to 255. Further and detailed information about the error source or location can be taken from the `Error_Code_1` and `Error_Code_2` byte. The explanation of the different errors can be found in the Chapter 'Error Response Definitions'.

4.2.6.10 MSAC2M_Read_reject

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF
	msg.ln	byte	8	length of message
	msg.nr	byte	CR	communication reference
	msg.a	byte	0x21	answer = MSAC2M_Read_Write_Data
	msg.f	byte	0x83 0x84	REJ_PS, REJ_LE
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0...126	Rem_Add
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0...254	Slot_Number
	msg.data_idx	byte	0...254	Index
	msg.data_cnt	byte	x 0...240	length of data block that could not be written
	msg.data_type	byte	10	data type, byte string = TASK_TDT_STRING
	msg.function	byte	1	function MSAC2M_Read = TASK_TFC_READ

If the requested write message contains inconsistencies then the command will be rejected.The explanation of the reject codes can be found in the Chapter 'Error Response Definitions'.

4.2.6.11 MSAC2M_Write_Req

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	3	receiver = USR_INTF
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	8+x	length of message
	msg.nr	byte	CR	communication reference
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0x21	command = MSAC2M_Read_Write_Data
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0...126	Rem_Add
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0...254	Slot_Number
	msg.data_idx	byte	0...254	Index
	msg.data_cnt	byte	x 0...240	length of data block to be written
	msg.data_type	byte	10	data type, byte string = TASK_TDT_STRING
	msg.function	byte	2	function MSAC2M_Write = TASK_TFC_WRITE
WRITE_RAW_DATA	msg.d[0]	byte		1'st data byte to slave
	...	...		...
	msg.d[x-1]	byte		last data byte to slave

The parameter `msg.device_adr = Rem_Add` contains the station address of the slave responder.
The parameter `msg.data_adr = Slot_Number` is used in the destination device for addressing the desired data block slot (typically a module).
The parameter `msg.data_idx = Index` is used in the destination device for addressing the desired data block.
The parameter `msg.data_cnt = Length` indicates the number of bytes of the data block that has to be written. If the destination data block length is less than required then the reply will conatin an error message. If the data block length is greater or equal than the required length the reply conatins the number of bytes which have been written. The DP-Slave may answer with an error response if the data access is not allowed.
The field `msg.d[x]` contains the data block which has to be written and consists of the number of octets indicated in the length of the request.

4.2.6.12 MSAC2M_Write_con_pos

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF
	msg.ln	byte	8	length of message
	msg.nr	byte	CR	communication reference
	msg.a	byte	0x21	answer = MSAC2M_Read_Write_Data
	msg.f	byte	0	CON_POS
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0...126	Rem_Add
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0...254	Slot_Number
	msg.data_idx	byte	0...254	Index
	msg.data_cnt	byte	x 0...240	length of data block that was written
	msg.data_type	byte	10	data type, byte string = TASK_TDT_STRING
	msg.function	byte	2	function MSAC2M_Write = TASK_TFC_WRITE

If the slave has accepted the write access than the MSAC2M_Write_con_pos primitive is sent as answer message. `Msg.f` is equal 0 in this case.

4.2.6.13 MSAC2M_Write_con_neg

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF
	msg.ln	byte	8+2	length of message
	msg.nr	byte	CR	communication reference
	msg.a	byte	0x21	answer = MSAC2M_Read_Write_Data
	msg.f	byte	255	CON_NEG
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0...126	Rem_Add
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0...254	Slot_Number
	msg.data_idx	byte	0...254	Index
	msg.data_cnt	byte	x 0...240	length of data block that could not be written
	msg.data_type	byte	10	data type, byte string = TASK_TDT_STRING
	msg.function	byte	2	function MSAC2M_Write = TASK_TFC_WRITE
ERROR_CODE	msg.d[0]	byte		Error_Code_1
	msg.d[1]	byte		Error_Code_2

If the slave denies the write access the variable `msg.f` is set to 255. Further and detaild information about the error source or location can be taken from the `Error_Code_1` and `Error_Code_2` byte. The explanation of the different errors can be found in the Chapter 'Error Response Definitions'.

4.2.6.14 MSAC2M_Write_reject

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF
	msg.ln	byte	8	length of message
	msg.nr	byte	CR	communication reference
	msg.a	byte	0x21	answer = MSAC2M_Read_Write_Data
	msg.f	byte	0x83,0x84	REJ_LE, REJ_PS
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended Message Header	msg.device_adr	byte	0...126	Rem_Add
	msg.data_area	byte	0	data_area, unused
	msg.data_adr	word	0...254	Slot_Number
	msg.data_idx	byte	0...254	Index
	msg.data_cnt	byte	x 0...240	length of data block that could not be written
	msg.data_type	byte	10	data type, byte string = TASK_TDT_STRING
	msg.function	byte	2	function MSAC2M_Write = TASK_TFC_WRITE

If the requested write message contains inconsistencies then the command will be rejected.The explanation of the reject codes can be found in the Chapter 'Error Response Definitions'.

4.2.6.15 MSAC2M_Abort_ind

	receive message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF
	msg.ln	byte	4 or 6	length of message
	msg.nr	byte	CR	communication reference
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0x2F	command = MSAC2M_Abort
	msg.e	byte	0	extension
ABORT_REASON	msg.d[0]	byte	0=FALSE 1=TRUE	Remote_Generated Locally_Generated
	msg.d[1]	byte	0...255	Subnet
	msg.d[2]	byte	0...255	Reason_Code
	msg.d[3]	byte	0...126	Rem_Add
	msg.d[4...5]	word	0...65535	Additional_Detail if msg.ln = 6

A `MSAC2M_Abort_ind` indication at the HOST side can be received at any time as long as the connection is in established state. Getting an abort does not implicit mean that the connection is closed, it only serves to indicate to the HOST that the connection has been aborted, either locally or remote controlled. But each abort indication will close the connection afterwards automatically so that the HOST has to wait for the `MSAC2M_Closed_ind` primitive coming from the DEVICE afterwards also. Only the `MSAC2M_Closed_ind` declare the connection as closed and only this indication can be taken as trigger to initialize the connection again by the `MSAC2M_Initiate_req` primitive. An abort can be forced by the remote slave station or can be generated in case of communication errors for example locally. The reason why and the source who has generated the abort is handed over in the message too.

The variable Locally_Generated:
TRUE: the abort was generated locally
FALSE: the abort was generated by the remote station

The variable Subnet:
0: NO
1: SUBNET-LOCAL
2: SUBNET-REMOTE
3...255: reserved

The variable Reason_Code:

D7	D6	D5	D4	D3	D2	D1	D0
reserved = 0		Instance		Reason Code			

Instance: 0 = FDL, 1 = MSAC_C2, 2 = USER, 3 = Reserved.

If the Additional_Detail variable from a remote aborted connection is unequal 0 then the message contains this further abort information. The value is manufacturer specific.

Instance	Reason_Code	Name	Meaning
FDL	1	UE	FDL interface error
	2	RR	insufficient remote resource for that service
	3	RS	service at remote SAP not activated
	9	NR	response without response data
	10	DH	response sent with high priority level
	11	LR	no resource of local FDL
	12	RDL	response low but with neg acknowledge received
	13	RDH	response high bit with neg acknowledge received
	14	DS	master not in the logical ring
	15	NA	no response from the remote FDL
MSAC2	1	ABT_SE	sequence error, service not allowed in this state
	2	ABT_FE	invalid request PDU received
	3	ABT_TO	timeout of the connection
	4	ABT_RE	invalid response PDU received
	5	ABT_IV	invalid service from the HOST
	6	ABT_STO	Send_Timeout requested was to small
	7	ABT_IA	invalid additional address information
	8	ABT_OC	waiting for FDL_DATA_REPLY.con
USER	0...31		fully USER specific reason code

4.2.6.16 MSAC2M_Abort_req

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	3	receiver = USR_INTF
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	3	length of message
	msg.nr	byte	CR	communication reference
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0x2F	command = MSAC2M_Abort
	msg.e	byte	0	extension
ABORT_REASON	msg.d[0]	byte	0...255	Subnet
	msg.d[1]	byte	0...31	Reason_Code
REMOTE_ADDRESS	msg.d[2]	byte	0-126	Rem_Add

The `MSAC2M_Abort_req` primitive is used to abort a Class2 communication relationship. `msg.d[2]` must contain the remote address of that connection that shall be aborted.

The variable `Subnet`:
`0`: NO
`1`: SUBNET-LOCAL
`2`: SUBNET-REMOTE
`3...255`: reserved

The variable `Reason_Code`:

D7	D6	D5	D4	D3	D2	D1	D0
reserved = 0				Reason_Code			

The value of the `Reason_Code` is fully user specific.

After the `MSAC2M_Abort_req` primitive was requested the DEVICE will answer with a `MASC2M_Close_ind` to indicate that the connection has been closed.

4.2.6.17 MSAC2M_Close_ind

	receive message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF
	msg.ln	byte	1	length of message
	msg.nr	byte	CR	communication reference
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0x2E	command = MSAC2M_Close
	msg.e	byte	0	extension
REMOTE_ADDRESS	msg.d[0]	byte	0-126	Rem_Add

The `MSAC2M_Close_ind` indication declares a previous initialized Class2 connection as closed. No further messages like `MSAC2M_Abort_ind` must be awaited for any longer for this now closed connection. In

4.2.6.18 Error Response Definitions

The error coding scheme of DPV1 uses error codes for classification of the error and indication of an additional error code. Basic communication errors which are not related to the DPV1 specification are reported in `msg.f` of each answer message:

msg.f	signification	error source	help
0 = CON_OK	service could be executed without an error		
2 = CON_RR	resource unavailable	slave	slave has no left buffer space for the requested service
3 = CON_RS	requested function of master is not activated within the slave	slave	slave is not activated in its DPV1support
9 = CON_NR	no answer-data, although the slave has to reponse with data	slave	slave hasn't sent back any amount of data
17 = CON_NA	no response of the station	slave	check network wiring, check bus address of slave or baud rate support
18 = CON_DS	master not into the logical token ring	network in general	check master DP-Address or highest-station-Address of other masters. Examine bus wiring to bus short circuits.
25 = CON_NP	no plausible reaction of remote partner	slave	slave does not work DPV1 norm conform
54 = CON_AD	negative response received, acces denied	slave	access denied to requested data. Check Error_Code_1 and Error_Code_2 in reponse message to get closer error information
0x83 = REJ_PS	a previous service is still in process	HOST	wait for the outstanding answer first. Parallel services are not allowed
0x84 = REJ_LE	the length indicator msg.data_cnt oversteps maximum configured size	HOST	reduce length of message or enlarge maximum buffer size in SyCon or in slave data set

4.2.7 MPI

The MPI offers the following functions

MPI function overview		Remark
Connect	See remark for Read/Write	The connection is established automatically when using the first read/write command. The DEVICE remembers the initial parameter until the connection is disconnected.
Read/Write	Read and write Data Block (DB)	The connection is established automatically when using the first read/write command. The DEVICE remembers the initial parameter until the connection is disconnected.
	Read and write Memory (M)	
	Read and write IO (I and Q)	
	Read and write Counter (C)	
	Read and write Timer (T)	
Disconnect	MPI disconnect	-
OP Status	MPI Get OP Status	-

The MPI interface can handle one command (read or write) at a time per remote station only. This means that the HOST can activate one command message to the DEVICE and then has to wait for the answer message from the DEVICE before it is allowed to send the next command to this remote station again.

4.2.7.1 MPI Read and Write Data Block (DB)

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	8 9 - 224	length of message read access write access
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	0x31	command = MPI_Read_Write_DB
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	0-126	remote station address
	msg.data_area	byte	0-255	data area, high byte of offset adress in DB
	msg.data_adr	word	0-65534	data address, DB address
	msg.data_idx	byte	0-255	data index, low byte of offset address in DB
	msg.data_cnt	byte	1-216 1-222	data count, number of bytes to be written or to be read
	msg.data_type	byte	5	data type bytestring TASK_TDT_UINT8
	msg.function	byte	1 2	function code TASK_TFC_READ TASK_TFC_WRITE
WRITE_DATA	msg.d[0...(x-1)]	byte array		in write access data to be written

This command allows to read or write Data Registers within S7 components which are supporting the MPI protocol. This can be either done via the MPI interface if any is present or the PROFIBUS interface of the remote station. Normally the MPI interface is driven at 187,5kBaud but should be verified with the corresponding configuration tool before calling the command. Via the standard PROFIBUS line the baudrate is relevant which is configured for the whole PROFIBUS network. Ensure that all active (master) station have the same bus parameter settings, especially the Baudrate, the Highest Station Address and the Target Rotation Time.

If the MPI connection to the remote station is not established and the command is called, the DEVICE automatically establishes the connection in the background before accessing the registers.

The access is divided into 4 parameters. First the remote station address must be specified which is addressed during the access. It's either the MPI address of the remote component or its PROFIBUS address. It depends which physical interface is used for the connection. The value must be fixed in `msg.device_adr`. A

range from 0 to 126 is possible. As next the DB value must be set in `msg.data_adr`. Allowed values are 0 to 65534. Within the DB there can be an offset specified from where the data is read or where the data is written to. The offset is splitted into 2 values. The offset is caculated by offset = `msg.data_area` * 256 + `msg.data_idx`. With that a range of 0 to 65534 is possible. The `msg.data_cnt` is the number of bytes to be read or written. The maximum value here is 222 bytes for read respectively 216 bytes for write. The value in `msg.function` specifies a read or write command.

	answer message			
Message header	variable	type	value	signification
	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF-Task
	msg.ln	byte	9-230 8	length of message read access write access
	msg.nr	byte	j	number of the message
	msg.a	byte	0x31	answer = MPI_Read_Write_DB
	msg.f	byte	f	error, see chapter error definitions in MPI
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	0-126	remote station address
	msg.data_area	byte	0-255	data area, high byte of offset adress in DB
	msg.data_adr	word	0-65534	data address, DB address
	msg.data_idx	byte	0-255	data index, low byte of offset address in DB
	msg.data_cnt	byte	1-216 1-222	data count, number of bytes which were written or which are read
	msg.data_type	byte	5	data type bytestring TASK_TDT_UINT8
	msg.function	byte	1 2	function code TASK_TFC_READ TASK_TFC_WRITE
READ_DATA	msg.d[0...(x-1)]	byte array		in read access data which was read

In case of a read command the answer message contains in `msg.d[...]` area the data from the DB. In case of a write command just the extended message is delivered back. If an error happened during the access the variable `msg.f` contains a value unequal 0. The definitions of the error codes can be read in the chapter 'Error code definition in MPI response messages'.

4.2.7.2 MPI Read and Write Memory (M)

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	 8 9 - 224	length of message read access write access
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	0x33	command = MPI_Read_Write_M
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	0-126	remote station address
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	 0-65534	data address, Memory address
	msg.data_idx	byte	0	data index, unsued
	msg.data_cnt	byte	 1-216 1-222	data count, number of bytes to be written or to be read
	msg.data_type	byte	 5	data type bytestring TASK_TDT_UINT8
	msg.function	byte	 1 2	function code TASK_TFC_READ TASK_TFC_WRITE
WRITE_DATA	msg.d[0...(x-1)]	byte array		in write access data to be written

This command allows to read or write Merker Registers within S7 components which are supporting the MPI protocol. This can be either done via the MPI interface if any is present or the PROFIBUS interface of the remote station. Normally the MPI interface is driven at 187,5kBaud but should be verified with the corresponding configuration tool before calling the command. Via the standard PROFIBUS line the baudrate is relevant which is configured for the whole PROFIBUS network. Ensure that all active (master) station have the same bus parameter settings, especially the Baudrate, the Highest Station Address and the Target Rotation Time.
If the MPI connection to the remote station is not established and the command is called, the DEVICE automatically establishes the connection in the background before accessing the registers.
The access is divided into 3 parameter. First the remote station address must be specified which is addressed during the access. It's either the MPI address of the remote component or its PROFIBUS address. It depends which physical interface is used for the connection. The value must be fixed in `msg.device_adr`. A range from 0 to 126 is possible. As next the Merkerbyte offset value must be set in `msg.data_adr`. Allowed values are 0 to 65534. The `msg.data_cnt` is the number of bytes to be read or written. The maximum value here is 222 bytes

for read respectively 216 bytes for write. The value in `msg.function` specifies a read or write command.

	answer message			
Message header	variable	type	value	signification
	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF-Task
	msg.ln	byte	9-230 8	length of message read access write access
	msg.nr	byte	j	number of the message
	msg.a	byte	0x33	answer = MPI_Read_Write_M
	msg.f	byte	f	error, see chapter error definitions in MPI
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	0-126	remote station address
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	0-65534	data address, Merker address
	msg.data_idx	byte	0	data index, unused
	msg.data_cnt	byte	1-216 1-222	data count, number of bytes which were written or which are read
	msg.data_type	byte	5	data type bytestring TASK_TDT_UINT8
	msg.function	byte	1 2	function code TASK_TFC_READ TASK_TFC_WRITE
READ_DATA	msg.d[0...(x-1)]	byte array		in read access data which was read

In case of a read command the answer message contains in `msg.d[...]` area the data from the Merker area. In case of a write command just the extended message header is delivered back. If an error happened during the access the variable `msg.f` contains a value unequal 0. The definitions of the error codes can be read in the chapter 'Error code definition in MPI response messages'.

4.2.7.3 MPI Read and Write IO (I and Q)

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	 8 9 - 224	length of message read access write access
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	0x34	command = MPI_Read_Write_IO
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	0-126	remote station address
	msg.data_area	byte	 0 1	data area, input area output area
	msg.data_adr	word	 0-65534	data address, IO address
	msg.data_idx	byte	0	data index, unsued
	msg.data_cnt	byte	 1-216 1-222	data count, number of bytes to be written or to be read
	msg.data_type	byte	 5	data type bytestring TASK_TDT_UINT8
	msg.function	byte	 1 2	function code TASK_TFC_READ TASK_TFC_WRITE
WRITE_DATA	msg.d[0...(x-1)]	byte array		in write access data to be written

This command allows to read or write IO Registers within S7 components which are supporting the MPI protocol. This can be either done via the MPI interface if any is present or the PROFIBUS interface of the remote station. Normally the MPI interface is driven at 187,5kBaud but should be verified with the corresponding configuration tool before calling the command. Via the standard PROFIBUS line the baudrate is relevant which is configured for the whole PROFIBUS network. Ensure that all active (master) station have the same bus parameter settings, especially the Baudrate, the Highest Station Address and the Target Rotation Time.

If the MPI connection to the remote station is not established and the command is called, the DEVICE automatically establishes the connection in the background before accessing the registers.

The access is divided into 4 parameters. First the remote station address must be specified which is addressed during the access. It's either the MPI address of the remote component or its PROFIBUS address. It depends which physical interface is used for the connection. The value must be fixed in `msg.device_adr`. A range from 0 to 126 is possible. The parameter `msg.data_area` selects the input area = 0 or the output area = 1. As next the IO-Byte offset addess must be set in `msg.data_adr`. Allowed values are 0 to 65534. The `msg.data_cnt` is the number of bytes to be read or written. The maximum value here is 222 bytes for read respectively 216 bytes for write. The value in `msg.function` specifies a read or write command.

	answer message			
Message header	variable	type	value	signification
	msg.rx	byte	16	transmitter = user at HOST
	msg.tx	byte	3	receiver = USR_INTF-Task
	msg.ln	byte	 9-230 8	length of message read access write access
	msg.nr	byte	j	number of the message
	msg.a	byte	0x34	answer = MPI_Read_Write_IO
	msg.f	byte	f	error, see chapter error definitions in MPI
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	0-126	remote station address
	msg.data_area	byte	 0 1	data area, input area output area
	msg.data_adr	word	 0-65534	data address, IO address
	msg.data_idx	byte	0	data index, unused
	msg.data_cnt	byte	 1-216 1-222	data count, number of bytes which were written or which are read
	msg.data_type	byte	 5	data type bytestring TASK_TDT_UINT8
	msg.function	byte	 1 2	function code TASK_TFC_READ TASK_TFC_WRITE
READ_DATA	msg.d[0...(x-1)]	byte array		in read access data which was read

In case of a read command the answer message contains in `msg.d[...]` area the data from the I or O area. In case of a write command just the extended message header is delivered back. If an error happened during the access the variable `msg.f` contains a value unequal 0. The definitions of the error codes can be read in the chapter 'Error code definition in MPI response messages'.

4.2.7.4 MPI Read and Write Counter (C)

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	8 10 - 224	length of message read access write access
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	0x35	command = MPI_Read_Write_Cnt
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	0-126	remote station address
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	0-65534	data address, counter address
	msg.data_idx	byte	0	data index, unsued
	msg.data_cnt	byte	1-108 1-111	data count, number of counters to be written or to be read
	msg.data_type	byte	6	data type word TASK_TDT_UINT16
	msg.function	byte	1 2	function code TASK_TFC_READ TASK_TFC_WRITE
WRITE_DATA	msg.d[0...(x-1)]	word array		in write access data to be written

This command allows to read or write Counter Registers within S7 components which are supporting the MPI protocol. This can be either done via the MPI interface if any is present or the PROFIBUS interface of the remote station. Normally the MPI interface is driven at 187,5kBaud but should be verified with the corresponding configuration tool before calling the command. Via the standard PROFIBUS line the baudrate is relevant which is configured for the whole PROFIBUS network. Ensure that all active (master) station have the same bus parameter settings, especially the Baudrate, the Highest Station Address and the Target Rotation Time.
If the MPI connection to the remote station is not established and the command is called, the DEVICE automatically establishes the connection in the background before accessing the registers.
The access is divided into 3 parameters. First the remote station address must be specified which is addressed during the access. It's either the MPI address of the remote component or its PROFIBUS address. It depends which physical interface is used for the connection. The value must be fixed in `msg.device_adr`. A range from 0 to 126 is possible. As next the Counter start offset addess must be set in `msg.data_adr`. Allowed values are 0 to 65534. The `msg.data_cnt` is the number of conters to be read or written. The maximum value here is 111

counters for read respectively 108 counters for write. The value in `msg.function` specifies a read or write command.

	answer message			
Message header	variable	type	value	signification
	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF-Task
	msg.ln	byte	 9-230 8	length of message read access write access
	msg.nr	byte	j	number of the message
	msg.a	byte	0x35	answer = MPI_Read_Write_Cnt
	msg.f	byte	f	error, see chapter error definitions in MPI
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	0-126	remote station address
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	 0-65534	data address, Counter address
	msg.data_idx	byte	0	data index, unused
	msg.data_cnt	byte	 1-108 1-111	data count, number of counter which were written or which are read
	msg.data_type	byte	 6	data type word TASK_TDT_UINT16
	msg.function	byte	 1 2	function code TASK_TFC_READ TASK_TFC_WRITE
READ_DATA	msg.d[0...(x-1)]	word array		in read access counter data which was read

In case of a read command the answer message contains in `msg.d[...]` area the data from the counter(s). In case of a write command just the extended message header is delivered back. If an error happened during the access the variable `msg.f` contains a value unequal 0. The definitions of the error codes can be read in the chapter 'Error code definition in MPI response messages'.

4.2.7.5 MPI Read and Write Timer (T)

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	8 10 - 224	length of message read access write access
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	0x36	command = MPI_Read_Write_Tim
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	0-126	remote station address
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	0-65534	data address, timer address
	msg.data_idx	byte	0	data index, unsued
	msg.data_cnt	byte	1-108 1-111	data count, number of timers to be written or to be read
	msg.data_type	byte	6	data type word TASK_TDT_UINT16
	msg.function	byte	1 2	function code TASK_TFC_READ TASK_TFC_WRITE
WRITE_DATA	msg.d[0...(x-1)]	word array		in write access data to be written

This command allows to read or write Timer Registers within S7 components which are supporting the MPI protocol. This can be either done via the MPI interface if any is present or the PROFIBUS interface of the remote station. Normally the MPI interface is driven at 187,5kBaud but should be verified with the corresponding configuration tool before calling the command. Via the standard PROFIBUS line the baudrate is relevant which is configured for the whole PROFIBUS network. Ensure that all active (master) station have the same bus parameter settings, especially the Baudrate, the Highest Station Address and the Target Rotation Time.
If the MPI connection to the remote station is not established and the command is called, the DEVICE automatically establishes the connection in the background before accessing the registers.
The access is divided into 3 parameters. First the remote station address must be specified which is addressed during the access. It's either the MPI address of the remote component or its PROFIBUS address. It depends which physical interface is used for the connection. The value must be fixed in `msg.device_adr`. A range from 0 to 126 is possible. As next the Timer start offset addess must be set in `msg.data_adr`. Allowed values are 0 to 65534. The `msg.data_cnt` is the number of timers to be read or written. The maximum value here is 111 timers

for read respectively 108 timers for write. The value in `msg.function` specifies a read or write command.

	answer message			
Message header	variable	type	value	signification
	msg.rx	byte	16	transmitter = user at HOST
	msg.tx	byte	3	receiver = USR_INTF-Task
	msg.ln	byte	 9-230 8	length of message read access write access
	msg.nr	byte	j	number of the message
	msg.a	byte	0x36	answer = MPI_Read_Write_Tim
	msg.f	byte	f	error, see chapter error definitions in MPI
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	0-126	remote station address
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	 0-65534	data address, Merker address
	msg.data_idx	byte	0	data index, unused
	msg.data_cnt	byte	 1-108 1-111	data count, number of timers which were written or which are read
	msg.data_type	byte	 6	data type word TASK_TDT_UINT16
	msg.function	byte	 1 2	function code TASK_TFC_READ TASK_TFC_WRITE
READ_DATA	msg.d[0...(x-1)]	word array		in read access counter data which was read

In case of a read command the answer message contains in `msg.d[...]` area the data from the timer(s). In case of a write command just the extended message header is delivered back. If an error happened during the access the variable `msg.f` contains a value unequal 0. The definitions of the error codes can be read in the chapter 'Error code definition in MPI response messages'.

4.2.7.6 MPI Disconnect

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	8	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	0x3F	command = MPI_Disconnect
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	0-126	remote station address
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	0	data address, unused
	msg.data_idx	byte	0	data index, unused
	msg.data_cnt	byte	0	data count , unused
	msg.data_type	byte	0	data type, unused
	msg.function	byte	0	function code, unused

While the DEVICE automatically establishes the connection to the remote station if not connected previously to it, the disconnect command must be executed by the HOST application.
This service should be executed after having finished the access to the station to close the connection properly and the reserved communication channels (SAPs) are freed again.

	answer message			
Message header	variable	type	value	signification
	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF-Task
	msg.ln	byte	8	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0x3F	answer = MPI_Disconnect
	msg.f	byte	f	error, see chapter error definitions in MPI
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	0-126	remote station address
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	0	data address, unused
	msg.data_idx	byte	0	data index, unused
	msg.data_cnt	byte	0	data count , unused
	msg.data_type	byte	0	data type, unused
	msg.function	byte	0	function code, unused

4.2.7.7 MPI Get OP Status

	command message			
Message header	variable	type	value	signification
	msg.rx	byte	3	receiver = USR_INTF-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	8	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	0x32	command = MPI_Get_OP_Status
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	0-126	remote station address
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	0	data address, unused
	msg.data_idx	byte	0	data index, unused
	msg.data_cnt	byte	0	data count , unused
	msg.data_type	byte	0	data type, unused
	msg.function	byte	0	function code, unused

This service provides the possibiltiy to read out the current operational status of the remote station.The remote address must be set in `msg.device_adr`. If an error happened during the access the variable `msg.f` contains a value unequal 0. The definitions of the error codes can be read in the chapter 'Error code definition in MPI response messages'.

	answer message			
Message header	variable	type	value	signification
	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = USR_INTF-Task
	msg.ln	byte	10	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0x32	answer = MPI_Get_OP_Status
	msg.f	byte	f	error, see chapter error definitions in MPI
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Extended message header	msg.device_adr	byte	0-126	remote station address
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	0	data address, unused
	msg.data_idx	byte	0	data index, unused
	msg.data_cnt	byte	0	data count , unused
	msg.data_type	byte	0	data type, unused
	msg.function	byte	0	function code, unused
OP_STATUS	msg.d[0-1]	word		operational status 0 STOP 1 START 2 RUN 3 UNKNOWN

4.2.7.8 Error Codes Definitions in MPI Response Messages

error code	signification	error source	help
0 = CON_OK	service could be executed without an error		
1 = CON_UE	timeout from remote station	remote station	remote station has not responded within 1 sec.timeout
2 = CON_RR	resource unavailable	remote station	remote station has no left buffer space for the requested service
3 = CON_RS	requested function of master is not activated within the remote station.	remote station	the connection seems to be closed in the remote station.try to send command again.
17 = CON_NA	no response of the remote station	remote station	check network wiring, check remote address, check baud rate
18 = CON_DS	master not into the logical token ring	network in general	check master DP-Address or highest-station-Address of other masters. Examine bus wiring to bus short circuits.
20 = CON_LR	Resource of the local FDL controller not available or not sifficient.	HOST	too many messages. no more segments in DEVICE free
21 = CON_IV	the specified msg.data_cnt parameter invalid	HOST	check the limit of 222 bytes (read) respectively 216 bytes (write) in msg.data_cnt
48 = CON_TO	timeout, the request message was accepted but no indication is sent back by the remote station	remote station	MPI protocol error, or station not presentor
57 = CON_SE	Sequence fault, internal state machine error. Remote station does not react like awaited or a reconnection was retried while connection is already open or device has no SAPs left to open connection channel	remote station	in case of sequence fault consult support center else retry request service again
0x85 = REJ_IV	specified offset address out of limits or not known in the remote station	HOST	please check msg.data_adr if present or offset parameter in request message
0x86 = REJ_PDU	wrong PDU coding in the MPI response of the remote station	DEVICE	contact hotline
0x87 = REJ_OP	specified length to write or to read results in an access outside the limits	HOST	please check msg.data_cnt length in request message

4.3 The FDL-Task

The following subsections abstractly describe the data transfer (FDL = Fieldbus Data Link) services of the EN 50170 Volume 2 bus. The FDL services are made available to the HOST via Layer 2.

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

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

Send Data with Acknowledge (SDA)
This service allows the HOST of the FDL (Layer 2) in the DEVICE (called local User in the following), to send user data (L_sdu) to a single remote station. At the remote station the L_sdu, if received error-free, is delivered by its FDL to the HSOT (called Remote User in the following). The Local HOST receives a confirmation concerning the receipt or non-receipt of the user data. If an error occured during the transfer, the FDL of the Local HOST shall repeat the data transfer.

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

Send and Request with Reply (SRD)
This service allows a Local HOST to transfer data (L_sdu) to a single remote station and at the same time to request data (L_sdu) that was made available by the Remote HOST at an earlier time. At the remote station the received L_sdu, if error-free, is passed to the Remote HOST. The service also allows a Local HOST to request data from the Remote HOST without sending data (L_sdu = Null) to the Remote HOST.
The Local HOST receives either the requested data or an indication that the requested data was not available or a confirmation of the non-receipt of the transmitted data. The first two reactions also confirm the receipt of the transferred data.

Sending all these 3 FDL commands as local station does not need any activation within the DEVICE. But to activate and to configure the DEVICE for reception of an SDA and SDN service as a remote station, use the service SAP activate. To activate and to configure the DEVICE for reception of an SRD service as remote station, use the RSAP activate service.

4.3.1 FDL_Send_Data_Ack_Req/Con / SDA

This command allows to send a SDA frame to the PROFIBUS network with variable data length field transparently. At the remote station the user data is delivered to the user. The local confirmation message of this command informs about the receipt or non-receipt of the user data in the remote station.

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	7	receiver = FDL-Task
	msg.tx	byte	16	transmitter = HOST
	msg.ln	byte	x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0x80	command = FDL_Send_Data_Ack_Req
	msg.e	byte	0	extension
FDL_DATA_ACK_REQ_STU	msg.d[0...(x-1)]	struct		

Here is the definition of the

```
typedef struct {
  unsigned char bLen; /*length of data field 0-244 */
  unsigned char bValue[ DATA_STU_SIZE ]; /* data */
} octet_str;

#define DATA_STU_SIZE  244

typedef struct FDL_DATA_ACK_REQ_STUtag{
  unsigned char bSsap;/*Src.Serv.Acc.Pt.0-62,255 */
  unsigned char bDsap;/*Dest.Serv.Acc.Pt.0-63,255 */
  unsigned char bRem_add_da;/*Remote Address 0-126 */
  unsigned char bServ_class;/* Prio Low=0,high=1*/
  octet_str tL_sdu;   /* data field structure */
} FDL_DATA_ACK_REQ_STU;
```

The parameter `bSsap` defines the service access point of the Local User.A value `bSsap` of 63 which is the global access address is not permissible in this command. Value 255 is the NIL SAP.
The parameter `bDsap` defines the service access point of the Remote User. Value 255 is the NIL SAP.
The parameter `bRem_add_da` defines the FDL address of the remote station.The remote address `bRem_add_da` 127 which is the global address is not permissible in this command.
The parameter `bServ_class` defines the FDL priority for the data transfer. Two priorities are possible `LOW = 0` and `HIGH = 1`.

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = HOST
	msg.tx	byte	7	transmitter = FDL-Task
	msg.ln	byte	5	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0x80	answer = FDL_Send_Data_Ack_Con
	msg.f	byte	f	error, see table below
	msg.b	byte	0	no command
	msg.e	byte	0	extension
FDL_DATA_ACK_CON_STU	msg.d[0...4]	struct		

```
typedef struct {
  unsigned char bSsap;/*Src.Serv.Acc.Pt.0-62,255 */
  unsigned char bDsap;/*Dest.Serv.Acc.Pt.0-63,255 */
  unsigned char bRem_add_da;/*Remote Address 0-126 */
  unsigned char bServ_class;/* Prio Low=0,high=1*/
  unsigned char bL_status /* error code */
} FDL_DATA_ACK_CON_STU;
```

Here is the definition of the possible `bL_status` error codes that can be received.

bL_status	signification	error source	help
0 = CON_OK	service could be executed without an error		
2 = CON_RR	resource unavailable	slave	slave has no left buffer space for the requested service
3 = CON_RS	requested function of master is not activated within the slave	slave	slave is not supporting the requested service
9 = CON_NR	no answer-data, although the slave has to reponse with data	slave	slave hasn't sent back any amount of data
17 = CON_NA	no response of the station	slave	check network wiring, check bus address of slave or baud rate support
18 = CON_DS	master not into the logical token ring	network in general	check master DP-Address or highest-station-Address of other masters. Examine bus wiring to bus short circuits.

4.3.2 FDL_Send_Data_Ack_Ind / SDA

This command is passed from the remote FDL controller to the local HOST after receipt of a SDA frame, if the acknowlegdement frame was sent. The receiption of the indication message needs no acknowledgment message to the local FDL.

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = HOST
	msg.tx	byte	7	transmitter = FDL-Task
	msg.ln	byte	x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0xC0	command = FDL_Send_Data_Ack_Ind
	msg.e	byte	0	extension
FDL_DATA_ACK_IND_STU	msg.d[0...(x-1)]	struct		

Here is the definition of the

```
typedef struct {
   unsigned char bLen;/*length of data field 0-244 */
    unsigned char bValue[ DATA_STU_SIZE ];/* data */
} octet_str;

#define DATA_STU_SIZE  244

typedef struct FDL_DATA_ACK_IND_STUtag{
  unsigned char bDsap;/*Dest.Serv.Acc.Pt.0-63,255*/
  unsigned char bSsap;/*Src.Serv.Acc.Pt.0-62,255 */
  unsigned char bRem_add_da;/*Remote Address 0-126 */
  unsigned char bLoc_add_sa;/*Local Address 0-126 */
  unsigned char bServ_class;/*Prio Low=0,high=1*/
  octet_str tL_sdu;   /*data field structure */
} FDL_DATA_ACK_IND_STU;
```

The parameter `bSsap` and `bDsap` specifiy the source and destination Service Access Points of the received SDA frame.
The parameter `bRem_add_da` defines the FDL address of the remote station which has sent this service.
The parameter `bServ_class` specifies the FDL priority of the received SDA frame.

4.3.3 FDL_Send_Data_No_Ack_Req / SDN

This command allows to send a SDN frame to the PROFIBUS network with variable data length field transparently to a single remote station, to many remote stations (multicast) or to all remote stations (broadcast) at the same time. The local confirmation message of this command informs just about the end of the transfer, but not that the data was duly received.

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	7	receiver = FDL-Task
	msg.tx	byte	16	transmitter = HOST
	msg.ln	byte	x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	0x81	command = FDL_Send_Data_No_Ack_Req
	msg.e	byte	0	extension
FDL_DATA_ACK_REQ_STU	msg.d[0...(x-1)]	struct		

Here is the definition of the

```
typedef struct octet_strtag{
  unsigned char bLen;/*length of data field 0-244*/
   unsigned char bValue[ DATA_STU_SIZE ]; /*data*/
} octet_str;

#define DATA_STU_SIZE  244

typedef struct FDL_DATA_ACK_REQ_STUtag {
  unsigned char bSsap;/* Src.Serv.Acc.Pt. 0-62,255 */
  unsigned char bDsap;/*Dest.Serv.Acc.Pt. 0-63,255 */
  unsigned char bRem_add_da;/*Remote Address 0-126*/
  unsigned char bServ_class;/*Prio Low=0,high=1*/
  octet_str tL_sdu;   /*data field structure*/
} FDL_DATA_ACK_REQ_STU;
```

The parameter `bSsap` defines the service access point of the Local User.A value `bSsap` of 63 which is the global access address is not permissible in this command.Value 255 is the NIL SAP.
The parameter `bDsap` defines the service access point of the Remote User. For broadcast messages a `bDsap` of 63 shall be chosen. In case of multicast messages the selection (group of stations) is performed by means of a dedicated `bDsap`.Value 255 is the NIL SAP.
The parameter `bRem_add_da` defines the FDL address of the remote station.
The parameter `bServ_class` defines the FDL priority for the data transfer. Two priorities are possible `LOW = 0` and `HIGH = 1`.

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = HOST
	msg.tx	byte	7	transmitter = FDL-Task
	msg.ln	byte	5	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0x81	answer = FDL_Send_Data_No_Ack
	msg.f	byte	f	error, see table below
	msg.b	byte	0	no command
	msg.e	byte	0	extension
FDL_DATA_ACK_CON_STU	msg.d[0...4]	struct		

```
typedef struct FDL_DATA_ACK_CON_STUtag {
  unsigned char bSsap;/* Src.Serv.Acc.Pt. 0-62 */
  unsigned char bDsap;/* Dest.Serv.Acc.Pt. 0-63 */
  unsigned char bRem_add_da;/*Remote Address 0-126*/
  unsigned char bServ_class;/*Prio Low=0,high=1*/
  unsigned char bL_status /*error code*/
} FDL_DATA_ACK_CON_STU;
```

Here is the definition of the possible `bL_status` error codes that can be received.

bL_status	signification	error source	help
0 = CON_OK	service could be executed without an error		
18 = CON_DS	master not into the logical token ring	network in general	check master DP-Address or highest-station-Address of other masters. Examine bus wiring to bus short circuits.

4.3.4 FDL_Send_Data_No_Ack_Ind / SDN

This command is passed from the remote FDL controller to the local HOST after receipt of a SDN frame, if the acknowlegdement frame was sent. The receiption of the indication message needs no acknowledgment message to the local FDL.

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = HOST
	msg.tx	byte	7	transmitter = FDL-Task
	msg.ln	byte	x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0xC1	command = FDL_Send_Data_No_Ack_Ind
	msg.e	byte	0	extension
FDL_DATA_ACK_IND_STU	msg.d[0...(x-1)]	struct		

Here is the definition of the

```
typedef struct {
  unsigned char bLen;/*length of data field 0-244*/
   unsigned char bValue[ DATA_STU_SIZE ]; /*data*/
} octet_str;

#define DATA_STU_SIZE  244

typedef struct FDL_DATA_ACK_IND_STUtag{
  unsigned char bDsap;/*Dest.Serv.Acc.Pt.0-63,255*/
   unsigned char bSsap;/*Src.Serv.Acc.Pt.0-62,255*/
  unsigned char bRem_add_da;/*Remote Address 0-126*/
  unsigned char bLoc_add_sa;/* Local Address 0-126*/
  unsigned char bServ_class;/* Prio Low=0,high=1*/
  octet_str tL_sdu;   /*data field structure*/
} FDL_DATA_ACK_IND_STU;
```

The parameter `bSsap` and `bDsap` specifiy the source and destination Service Access Points of the received SDN frame.
The parameter `bRem_add_da` defines the FDL address of the remote station which has sent this service.
The parameter `bServ_class` specifies the FDL priority of the received SDN frame.

4.3.5 FDL_Send_Data_With_Reply_Req/Con / SRD

This command allows to transfer data to a single remote station and at the same time to request data that was made available by the remote user at an earlier time. The data of the response frame is sent back in the response message transparently. The service allows also to request data from the remote station without sending data to the remote station.(L_sdu = empty).

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	7	receiver = FDL-Task
	msg.tx	byte	16	transmitter = HOST
	msg.ln	byte	x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	0x82	command = FDL_Send_Data_Reply
	msg.e	byte	0	extension
FDL_DATA_REPLY_REQ_STU	msg.d[0...(x-1)]	struct		

Here is the definition of the structures:

```
typedef struct octet_strtag{
   unsigned char bLen; /*length of data field 0-244*/
    unsigned char bValue[ DATA_STU_SIZE ]; /*data*/
} octet_str;

#define DATA_STU_SIZE  244

typedef struct FDL_DATA_REPLY_REQ_STUtag {
  unsigned char bSsap;/*Src.Serv.Acc.Pt. 0-62,255*/
  unsigned char bDsap;/*Dest.Serv.Acc.Pt. 0-62,255*/
  unsigned char bRem_add_da;/*Remote Address 0-126*/
  unsigned char bServ_class;/*Prio Low=0,high=1*/
  octet_str tL_sdu;   /*data field structure*/
} FDL_DATA_REPLY_REQ_STU;
```

The parameter `bDsap` defines the service access point of the Remote User. Value 255 is the NIL SAP.
The parameter `bSsap` defines the service access point of the Local User.A value `bSsap` of 63 which is the global access address is not permissible in this command.Value 255 is the NIL SAP.
The parameter `bRem_add_da` defines the FDL address of the remote station.
The parameter `bServ_class` defines the FDL priority for the data transfer. Two priorities are possible `LOW = 0` and `HIGH = 1`.

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = HOST
	msg.tx	byte	7	transmitter = FDL-Task
	msg.ln	byte	x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0x82	answer = FDL_Send_Data_Reply
	msg.f	byte	f	error, see table below
	msg.b	byte	0	no command
	msg.e	byte	0	extension
FDL_DATA_REPLY_CON_STU	msg.d[0...(x-1)]	struct		

```
typedef struct FDL_DATA_REPLY_CON_STUtag {
  unsigned char bSsap;/*Src.Serv.Acc.Pt. 0-62,255 */
  unsigned char bDsap;/*Dest.Serv.Acc.Pt. 0-62,255 */
  unsigned char bRem_add_da;/*Remote Address 0-126 */
  unsigned char bServ_class;/*Prio Low=0,high=1*/
  unsigned char bL_status /*error code*/
  octet_str tL_sdu;   /*data field structure*/
} FDL_DATA_REPLY_CON_STU;
```

The `tL_sdu` data field contains only valid data, if the `bL_status` variable is 0. Here is the definition of the possible `bL_status` error codes that can be received.

bL_status	signification	error source	help
0 = CON_OK	service could be executed without an error		
2 = CON_RR	resource unavailable	slave	slave has no left buffer space for the requested service
3 = CON_RS	requested function of master is not activated within the slave	slave	slave is not supporting the requested service
8 = CON_DL	response data available, sent as low telegram	no error	the slave has responded with data in a low priority telegram
9 = CON_NR	no answer-data, although the slave has to reponse with data	slave	slave hasn't sent back any amount of data
17 = CON_NA	no response of the station	slave	check network wiring, check bus address of slave or baud rate support
18 = CON_DS	master not into the logical token ring	network in general	check master DP-Address orhighest-station-Address of other masters. Examine bus wiring to bus short circuits.

4.3.6 FDL_Send_Data_With_Reply_Ind / SRD

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = HOST
	msg.tx	byte	7	transmitter = FDL-Task
	msg.ln	byte	x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	f	error, see table below
	msg.b	byte	0xC2	command = FDL_Send_Data_Reply_Ind
	msg.e	byte	0	extension
FDL_DATA_REPLY_IND_STU	msg.d[0...(x-1)]	struct		

```
typedef struct FDL_DATA_REPLY_IND_STUtag {
  unsigned char bDsap;/*Dest.Serv.Acc.Pt.0-62,255*/
   unsigned char bSsap;/*Src.Serv.Acc.Pt. 0-62,255*/
  unsigned char bRem_add_da;/*Remote Address 0-126*/
   unsigned char bLoc_add_da;/*local Address 0-126*/
 unsigned char bServ_class;/* Prio Low=0,high=1*/
 unsigned char bUpdate_Status/*status reply dat*/
 octet_str tL_sdu;   /*data field structure*/
} FDL_DATA_REPLY_IND_STU;
```

The parameter `bUpdate_Status` specifies, whether or not the response data has been passed to the local FDL controller. The response data can be set up to with the `FDL_Reply_Update` primitve. The following parameter values are defined:

```
#define CON_LO  0x20 /* low priority data transmitted
in response */
#define CON_HI  0x21 /* high priority data transmit-
ted in response */
#define CON_NO  0x22 /* no data transmitted in re-
sponse */
```

4.3.7 FDL_Reply_Update

The HOST is reponsible for valid data in the DEVICE is any data is shall be requested from a remote station. By this command the HOST may initiate the loading of this data to a specific SAP. Upon completion of loading the data area, the DEVICE informs the HOST by the confirmation message.

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	7	receiver = FDL-Task
	msg.tx	byte	16	transmitter = HOST
	msg.ln	byte	x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	0x83	command = FDL_Reply_Update_Req
	msg.e	byte	0	extension
FDL_REPLY_UPDATE_REQ _STU	msg.d[0...(x-1)]	struct		

Here is the definition of the structures:

```
typedef struct octet_strtag{
  unsigned char bLen;/*length of data field 0-244*/
   unsigned char bValue[ DATA_STU_SIZE ]; /*data*/
} octet_str;

#define DATA_STU_SIZE  244

typedef struct FDL_REPLY_UPDATE_REQ_STUtag {
  unsigned char bSsap;/*Src.Serv.Acc.Point 0-62,255*/
  unsigned char bDsap;/* not used */
  unsigned char bRem_add_da;/* not used */
  unsigned char bServ_class;/* Prio Low=0,high=1*/
  unsigned char bTransmit;/* Single=1/Multiple=2*/
  octet_str tL_sdu;   /* data field structure */
} FDL_REPLY_UPDATE_REQ_STU;
```

The parameter `bSsap` specifies the service access point of the remote User that carries out this request and which data area shall be updated by the tL_sdu. A bSsap valu of 63 is not permissible.
The parameter `bDsap` defines the service access point of the Remote User. Value 255 is the NIL SAP.
The parameter `bRem_add_da` defines the FDL address of the remote station.
The parameter `bServ_class` defines the FDL priority for the data transfer. Two priorities are possible `LOW = 0` and `HIGH = 1`.
The parameter `bTransmit` specifies whether the update is transmitted only once `SINGLE = 1` or many times `MULTIPLE = 2`, in the case of "multiple" the data area is transferred again for each subsequent SRD.

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = HOST
	msg.tx	byte	7	transmitter = FDL-Task
	msg.ln	byte	5	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0x83	answer = FDL_Reply_Update_Con
	msg.f	byte	f	error, see table below
	msg.b	byte	0	no command
	msg.e	byte	0	extension
FDL_REPLY_UPDATE_CON _STU	msg.d[0...4]	struct		

```
typedef struct FDL_REPLY_UPDATE_CON_STUtag{
  unsigned char bSsap;/*Src.Serv.Acc.Point 0-62*/
  unsigned char bDsap;/*Dest.Serv.Acc.Point 0-63*/
  unsigned char bRem_add_da;/*Remote Address 0-126*/
  unsigned char bServ_class;/*Prio Low=0,high=1*/
  unsigned char bL_status /*error code*/
} FDL_REPLY_UPDATE_CON_STU;
```

Here is the definition of the possible `bL_status` error codes that can be received.

bL_status	signification	error source	help
0 = CON_OK	service could be executed without an error		
18 = CON_DS	master not into the logical token ring	network in general	check master DP-Address or highest-station-Address of other masters. Examine bus wiring to bus short circuits.

4.3.8 FDL_Sap_Activate

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	7	receiver = FDL-Task
	msg.tx	byte	16	transmitter = HOST
	msg.ln	byte	x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0x97	command = FDL_Sap_Activate
	msg.e	byte	0	extension
FDL_SAP_ACTIVATE_REQ _STU	msg.d[0...(x-1)]	struct		

```
typedef struct FMA_SAP_ACTIVATE_REQ_STUtag {

  unsigned char bSsap; /*Src.Serv.Acc.Point 0-63,65*/
  unsigned char bAccess; /*Accessright = 255*/
  unsigned char bService_list_length;  /*Length = 4*/
  #define SERVICE_SDA  0
  #define SERVICE_SDN  1
  #define SERVICE_SRD  2
  unsigned char abService_activate[4]; /*Service*/
   #define ROLE_IN_BOTH      1
  #define ROLE_IN_RESPONDER 2
  #define ROLE_IN_INITIATOR 3
  unsigned char abRole_in_service [4]; /*Cfg.Srv.*/
  struct {
    unsigned char bReq_low;  /* bufferlength = 244 */
    unsigned char bReq_high; /* bufferlength = 244 */
    unsigned char bInd_low;  /* bufferlength = 244 */
    unsigned char bInd_high; /* bufferlength = 244 */
  } tNon_cyclic_L_sdu_length_list[3];

} FMA_SAP_ACTIVATE_REQ_STU;
```

This service provides the HOST with the possibility to activate and to configure a local LSAP for the individual FDL services. An exception to this is the responder function for the Reply services (SRD,CSRD) that are activated by means of the RSAP activate service.
The parameter `bSsap` specifies the local LSAP that is to be activated and configured. The values 0 to 63 and NIL = 65.
The parameter `bAccess` with the values ALL = 255 or 0 to 126 is used for access protection ans specifies whether all remote stations (all) or only an individual remote station (0 to 126) may have access to the LSAP. The parameter is only valid for responder functions, i.e. `abRole_in_service[...]` = `ROLE_IN_RESPONDER` / `ROLE_IN_BOTH`. If the access value is ALL, the DEVICE automatically sets the buffer length to the maximum value 244 and the

services to all supported and the role in service to both independant what the rest command message parameter are.

The parameters `abService_activate[...]` contains the FDL service that shall be activated for this service (SDA, SDN, SRD).
The parameters `abRole_in_service[...]` specifies the configuration for the service to be activated. The following values are specified:
`ROLE_IN_INITIATOR`: The DEVICE initiates the respective service exclusively
`ROLE_IN_RESPONDER`: The DEVICE reponds to the service exclusively.Not permissible for SRD.
`ROLE_IN_BOTH`: The DEVICE initiates and responds to the service. Not permissible for SRD.

The value `tNon_cyclic_L_sdu_length_list[...]`is a list of Max_L_sdu lengths. It specifies the maximum length low or high priority L_sdu for the requested primitive in `bReq_Low/High`. For the indication of the service SDA and SDN and for the confirmation of the service SRD, the maximum length is specified by the parameter `bInd_Low/High`.

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = HOST
	msg.tx	byte	7	transmitter = FDL-Task
	msg.ln	byte	x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0x97	answer = FDL_Sap_Activate
	msg.f	byte	0	no error
	msg.b	byte	0	no command
	msg.e	byte	0	extension
FDL_SAP_ACTIVATE_CON _STU	msg.d[0...(x-1)]	struct		

```
typedef struct FMA_SAP_ACTIVATE_CON_STUtag {
  unsigned char  bSsap;/*Src.Serv.Acc.Point 0-63,65*/
  unsigned char  bM_status; /* Error Code */
} FMA_SAP_ACTIVATE_CON_STU;
```

4.3.9 FDL_RSap_Activate

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	7	receiver = FDL-Task
	msg.tx	byte	16	transmitter = HOST
	msg.ln	byte	x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0x98	command = FDL_RSap_Activate
	msg.e	byte	0	extension
FDL_RSAP_ACTIVATE_REQ _STU	msg.d[0...(x-1)]	struct		

```
typedef struct FMA_RSAP_ACTIVATE_REQ_STUtag {
 unsigned char bSsap;  /*Src.Serv.Acc.Point 0-64,65*/
  unsigned char  bAccess; /*Accessright = 255*/
 unsigned char  bReq_low;  /* bufferlength = 240 */
 unsigned char  bReq_high; /* bufferlength = 240 */
 unsigned char  bInd_low;  /* bufferlength = 240 */
 unsigned char  bInd_high; /* bufferlength = 240 */
} FMA_RSAP_ACTIVATE_REQ_STU;
```

The parameter `bSsap` specifies the local LSAP for the data area and the user taht receives the Indication primitive if a SRD is received on this "DSAP".The value can have a range from 0 to 64 and NIL = 65.

The parameter `bAccess` with the values ALL = 255 or 0 to 126 is used for access protection ans specifies whether all remote stations (all) or only an individual remote station (0 to 126) may have access to the LSAP. The parameter is only valid for responder functions, i.e. `abRole_in_service[...]` = `ROLE_IN_RESPONDER` / `ROLE_IN_BOTH`.

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = HOST
	msg.tx	byte	7	transmitter = FDL-Task
	msg.ln	byte	x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0x98	answer = FDL_RSap_Activate
	msg.f	byte	0	no error
	msg.b	byte	0	no command
	msg.e	byte	0	extension
FDL_RSAP_ACTIVATE_CON_STU	msg.d[0...(x-1)]	struct		

```
typedef struct FMA_RSAP_ACTIVATE_CON_STUtag {
  unsigned char  bSsap;/*Src.Serv.Acc.Point 0-63,65*/
  unsigned char  bM_status; /* Error Code */
} FMA_RSAP_ACTIVATE_CON_STU;
```

4.3.10 FDL_Status_Reply

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	7	receiver = FDL-Task
	msg.tx	byte	16	transmitter = HOST
	msg.ln	byte	x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, state
	msg.b	byte	0x9B	command = FDL_Status_Reply
	msg.e	byte	0	extension
Message Data	msg.d[0..7]	byte array	0	reserved
	msg.d[8]	byte	0-126	bRem_add_da

This service serves to send the FDL service FDL Status with Reply and is used to get the current state of the specified Remote Station. The remote station itself must be set to byte `msg.d[8]` within the data are of the message and has a range of 0 to 126.

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = HOST
	msg.tx	byte	7	transmitter = FDL-Task
	msg.ln	byte	x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0x9B	answer = FDL_Status_Reply
	msg.f	byte	0	no error
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Message Data	msg.d[0..7]	byte array	0	reserved
	msg.d[8]	byte	0xff 0x30 0x00	status of each station: not available active station passive station

Error response defintions in `msg.f`:

msg.f	signification	error source	help
0 = CON_OK	service could be executed without an error		

18 = CON_DS	master not into the logical token ring	network in general	check master DP-Address or highest-station-Address of other masters. Examine bus wiring to bus short circuits.

4.3.11 FDL_Send_Time_Sync_Req / CS

This command allows to send a CS sequenz to the PROFIBUS network The local confirmation message of this command informs just about the end of the transfer, but not that the data was duly received.

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	7	receiver = FDL-Task
	msg.tx	byte	16	transmitter = HOST
	msg.ln	byte	x	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	0x89	command = FDL_Send_Data_No_Ack_Req
	msg.e	byte	0	extension
FDL_CS_REQ_STU	msg.d[0...(x-1)]	struct		

Here is the definition of the

```
typedef struct FDL_CS_REQ_STUtag{
  unsigned char bLen;/*length of data field 0-244*/
    unsigned long ulSecCnt;/*sec since the 1.1.1900*/
      unsigned long ul1_32usec;/* sec-part in 1/32
usec*/   unsigned char bValue[ DATA_STU_SIZE ]; /*da-
ta*/
} FDL_CS_REQ_STU;

#define DATA_STU_SIZE  236
```

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = HOST
	msg.tx	byte	7	transmitter = FDL-Task
	msg.ln	byte	5	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	0x89	answer = FDL_Send_Data_No_Ack
	msg.f	byte	f	error, see table below
	msg.b	byte	0	no command
	msg.e	byte	0	extension
FDL_CS_CON_STU	msg.d[0...4]	struct		

```
typedef struct FDL_DATA_ACK_CON_STUtag {
  unsigned char bL_status     /*error code*/
} FDL_DATA_ACK_CON_STU;
```

For the `bL_status` error the values OK, LS, LR, DS, SV and IV can be received.

5 Access of SyCon Configuration DBM-file using dbm32.dll

A `DBM` file, which can be created with SyCon option `Export` or with dbm32.dll function `DbmExportCfgToDbm()`, is a Hilscher defined data base format. Within such a data base different data base tables can exist. They are all distinguished by an ASCII character string of 32 bytes in maximal length. Each table can have at least one data set. If for example the bus parameter shall be accessed to, this table will consist of only one data set. The table where the configuration of the different bus devices is held will consist of one or more records.
The `DBM` file can be created based on a Windows ACCESS data base only, which is usually the configuration file itself. Of course the coded tables within the resulting `DBM` file will not contain every information that is held within the mother ACCESS data base.
Each `DBM` file has a unique 8 byte ASCII character string to distinguish each `DBM` file for each fieldbus which is held in the table `GLOBAL`. This character string is also stored in the DEVICE itself and is nameing the FLASH segment and is compared during the download procedure with the name of the `DBM` file. If both strings are matching, the download can be performed.

5.1 Getting the FLASH Segment, the Internal Name of the DBM file

Use the function `DbmGetFielbusName()` to get the name of the fieldbus.
For the Profibus PB DEVICES the following string will be returned:

HEX	50	42	5F	43	4F	4D	42	49
ASCII	P	B	_	C	O	M	B	I

For the Profibus DPM DEVICES the following string will be returned:

HEX	50	52	4F	46	49	42	55	53
ASCII	P	R	O	F	I	B	U	S

5.2 Getting and Accessing the Table Names

Use the function `DbmGetTableDesc()` to get an overview of all tables which are included in the file and the corresponding number of data sets per table.

The function will find the following table names:

HEX	47	4C	4F	42	41	4c	0
ASCII	G	L	O	B	A	L	

HEX	42	55	53	5F	44	50	0
ASCII	B	U	S	_	D	P	

HEX	53	4C	5F	44	50	0
ASCII	S	L	_	D	P	

HEX	50	4C	43	5F	50	41	52	41	4D	0
ASCII	P	L	C	_	P	A	R	A	M	

HEX	44	45	53	43	52	49	50	54	0
ASCII	D	E	S	C	R	I	P	T	

Each name is terminated with a 0 in the returned structure. For each table also a handle is returned which be must used for the later access to each table.

5.2.1 The relevant Tables and their Structure

Use the function `DbmGetRecords()` now to get the containments of each relevant table. If a table consist of more records, this function will return all records of this tables in one call.
After accessing the recordfile, be sure to free the allocated buffer memory with the function `DbmFreeBuffer()`.

5.2.1.1 The relevant Table BUS_DP, getting Bus Parameter Record

The table `BUS_DP` will consist of one record.
The pointer `pbRecordData` in the structure `REC` of the table BUS_DP points to a structure which is defined in the chapter 'Coding of the Bus Parameter Data Set'.

5.2.1.2 The relevant Table SL_DP, getting Slaves Parameter Records

The table SL_DP will consist of at least one record.Use the returned value lRecordStructCount to get the whole number of record entries of the tables SL_DP.
Each pointer pbRecordData in the structures REC points to a structure which is almost defined in the chapter 'Coding of the Slave Parameter Data Set'. But the structure described in this chapter is extended by one leading byte which is the configured Slave_Address.

Structure each pointer pbRecordData will be assigned to:

variable name	type	explanation
Slave_Address	**byte**	**Address of Slave 0-125**
Slave_Data_set	structure	Structure already defined in the chapter 'Coding of the Slave Parameter Data Set'.

6 General Procedure how to get the DEVICE operative without SyCon

Like in the chapters above described, the DEVICE supports the online configuration without using the SyCon configuration tool. That means the DEVICE must be initalized in its protocol parameters first (see. chapter protocol parameters), then a warmstart must be proceeded. After that the network specific parameter must be download via message functionality `Start_Seq, Download, End_Seq`. By using these functions, the network device specific parameters must be downloaded first and then the bus parameter must follow. The download of the bus parameter is the trigger point for the DEVICE to start its network activity the first time. Remember that these download parameter aren't stored in the DEVICE FLASH memory and are lost if the DEVICE is reseted or powered down.
The just described procedure does only work, if the DEVICE isn't configured by SyCon configuration tool, else the found FLASH configuration will always have higher priority then the HOST defined configuration download parameter. To ensure that the DEVICE itself is not preconfigured by SyCon with a static FLASH configuration, for example if you have receive a new delivered one, you have to proceed the following initial sequence to get every DEVICE working:

6.1 Using Device Driver Functions

1. `DevOpenDriver()`: Enable the link of the application to the device driver
2. `DevInitBoard()`: Link application to the specific DEVICE
3. `DevPutTaskParameter()`: Set up the protocol parameter
4. `DevReset(WARMSTART)`: Execute a warm start command to DEVICE
5. `DevGetBoardInfo(GET_DRIVER_INFO)`: Read driver state
6. Examine the variable `bHostFlags` in the backcoming driver state structure
7. If `bHostFlags` indicates the bits `RDY` and `RUN` then the DEVICE is configured by SyCon. Then execute Delete database message to DEVICE by using `DevPutMessage()` and `DevGetMessage()` procedure. Goto step 3 again
8. Use now `DevPutMessage()` and `DevGetMessage()` procedure to download the network specific configuration. After the download of the bus parameter data set the DEVICE automatically starts up the network.

6.2 Using direct access to the dual-port memory

1. Examine the cell `bHostFlags` directly. If cell indicates the bits `RDY` and `RUN` then the DEVICE is configured by SyCon. Then execute `delete database` message (see chapter in this manual) to DEVICE by using corresponding message algorithm described in the `toolkit general definitions` manual. Goto step 1. If cell indicates `RDY` only then goto step 2.
2. Write protocol specific parameter into corresponding `Task2Parameter` area.
3. Write WARMSTART = 0x40 into cell `bDevFlags` to execute the DEVICE's warmstart.
4. Now download the network specific configuration via message procedure. After the download of the bus parameter data set the DEVICE automatically starts up the network.

7 Technical Characteristics

7.1 Limiting Values

Limit description	limit value
number of DP-Slaves	127 (address range 0-126) Address 127 = broadcast address Address 126 = slave with changeable default address Address 0 = Master class 2 default address
number of process data per slave	244 input and 244 output bytes
max.number of process data DP-DEVICEs PB-DEVICEs	 512 byte input and 512 byte output 3584 byte input and 3584 byte output
max.number of diagnostics bytes per slave	100
max.number of configuration data bytes per slave	244
max.number of paramter data bytes per slave	244
max.number of maximum modules per modular slave	244
supported baud rates	9,6Kbaud;19,2kBaud;31,25kBaud;45,45kBaud;93,75KBaud;187,5Kbaud;500KBaud;1,5MBaud;3Mbaud;6Mbaud;12Mbaud
min. DP-scan cycletime	350µsec

7.2 Supported PROFIBUS-Services of DP-Master only DEVICEs

Service		DP-Master Class 1		DP-Master Class 2	
		Req	Res	Req	Res
Data-Exchange	Transfer Input and Output Data	✓			
RD_Inp	Read Input Data of a DP-Slave				
RD_Outp	Read Output Data of a DP-Slave				
Slave_Diag	Read DP-Slave Diagnostic Information	✓			
Set_Prm	Send Parameter Data	✓			
Chk_Cfg	Check Configuration Data	✓			
Get_Cfg	Read the Configuration Data			✓	
Global_Control	Control Commands to DP-Slave	✓			
Set_Slave_Add	Change Station Address of a DP-Slave			✓	
Get_Master_Diag	Read Master Diagnostic Information		✓		
Start_Seq	Download Parameter (start)	✓			
Download	Download Parameter	✓			
End_Seq	Download Parameter (end)	✓			

7.3 Supported PROFIBUS-Services of PB DEVICEs

Service		DP-Master Class 1		DP-Master Class 2	
		Req	Res	Req	Res
Data-Exchange	Transfer Input and Output Data	✓			
RD_Inp	Read Input Data of a DP-Slave			✓	
RD_Outp	Read Output Data of a DP-Slave			✓	
Slave_Diag	Read DP-Slave Diagnostic Information	✓			
Set_Prm	Send Parameter Data	✓			
Chk_Cfg	Check Configuration Data	✓			
Get_Cfg	Read the Configuration Data			✓	
Global_Control	Control Commands to DP-Slave	✓			
Set_Slave_Add	Change Station Address of a DP-Slave			✓	
Get_Master_Diag	Read Master Diagnostic Information		✓		
Start_Seq	Download Parameter (start)	✓			
Download	Download Parameter	✓			
End_Seq	Download Parameter (end)	✓			
Act_Para_Brct	Activate Parameter Set (unconfirmed)	✓			
Act_Param	Activate/Deactivate Parameter Set	✓			
MSAC1_Read	Read acyclic data	✓			
MSAC1_Write	Write acyclic data	✓			
MSAC1_Alarm	Send Slave Alarm to HOST	✓			
MSAC2_Read				✓	
MSAC2_Write				✓	
MSAC2_Initiate				✓	
MSAC2_Abort				✓	