SIEMENS

SINEC

DP Programming Interface

Volume 1 of 1

C79000-G8976-C071 Release 3

SIEMENS

Siemens Aktiengesellschaft

Elektronikwerk Karlsruhe
Printed in the Federal Republic of Germany

SIEMENS

SINEC
DP Programming Interface

Description C79000-B8976-C071/03

Note

We would point out that the contents of this product documentation shall not become a part of or modify any prior or existing agreement, commitment or legal relationship. The Purchase Agreement contains the complete and exclusive obligations of Siemens. Any statements contained in this documentation do not create new warranties or restrict the existing warranty.

We would further point out that, for reasons of clarity, these operating instructions cannot deal with every possible problem arising from the use of this device. Should you require further information or if any special problems arise which are not sufficiently dealt with in the operating instructions, please contact your local Siemens representative.

General

This device is electrically operated. In operation, certain parts of this device carry a dangerously high voltage.

WARNING !

!

Failure to heed warnings may result in serious physical injury and/or material damage.

Only appropriately qualified personnel may operate this equipment or work in its vicinity. Personnel must be thoroughly familiar with all warnings and maintenance measures in accordance with these operating instructions.

Correct and safe operation of this equipment requires proper transport, storage and assembly as well as careful operator control and maintenance.

Personnel qualification requirements

Qualified personnel as referred to in the operating instructions or in the warning notes are defined as persons who are familiar with the installation, assembly, startup and operation of this product and who posses the relevant qualifications for their work, e.g.:

– Training in or authorization for connecting up, grounding or labelling circuits and devices or systems in accordance with current standards in safety technology;

– Training in or authorization for the maintenance and use of suitable safety equipment in accordance with current standards in safety technology;

– First Aid qualification.

DP Programming Interface

The distributed I/Os (abbreviated to DP from now on) allow you to use a variety of analog and digital input/output modules with a distributed configuration in close proximity to the process.

There can be large distances between the individual I/O devices bridged by the serial field bus SINEC L2. Distributed I/O devices capture the input signals locally and transfer them via the field bus to the central controller in the PG/PC. In the opposite direction, the central controller sends output data to the distributed I/O devices cyclically.

Networking the components results in a considerable reduction in cabling compared with previous methods in which the components were “hard” wired.

The PROFIBUS DP protocol used for the distributed I/Os is based on the PROFIBUS DIN E 19245 Part 3 communications standard for the field area. The concept of DP communication was developed in a joint venture by leading manufacturers of programmable logic controllers. It describes a multivendor (heterogeneous) transmission protocol designed to meet the requirements of the field area. DP communication uses a subset of the open communications services standardized in DIN 19245 Part 1.

PROFIBUS DP is intended for time-critical applications. The simple, optimized transmission protocol, the high transmission rates and the use of a master-slave structure achieve short cycle times.

This volume describes the DP protocol and how to program it. ❑

NOTES

NOTES

1 Distributed I/Os

This chapter describes the basic characteristics of the distributed input/output system:

- ➢ Central control by a master
- ➢ High data throughput with a simple transmission protocol
- ➢ Cyclic transmission of the process image in the input/output direction
- ➢ Simple, cost-effective attachment
- ➢ Data transmission via twisted pair (RS 485) or optical fiber
- ➢ Detection of errors with on-line diagnostics
- ➢ Based on DIN 19245 Part 1, parallel operation with FMS devices (master and slaves) on one bus is possible.

I/O Devices from Siemens

A wide variety of I/O devices are available for various applications, for example:

ET 200U: A modular I/O device with the IP 20 degree of protection for universal application. The ET 200U system can be configured with up to 32 input/output modules.

ET 200B: A small compact I/O device with the IP 20 degree of protection. Various versions of the ET 200B system are available.

ET 200C: A robust I/O device with the IP 66/67 degree of protection for use in a hostile industrial environment.

Design and Installation

For detailed information about the functions, design and installation of the I/O devices listed above, refer to the manuals for the particular product.

Further Information

Further Information about available components and the connection of devices can be found in the SINEC Catalog IK 10.

1.1 DP Protocol

Basic Design

Fig. 1.1 shows the basic design and components of a SINEC L2 DP system controlled by one computer with a PROFIBUS CP installed.

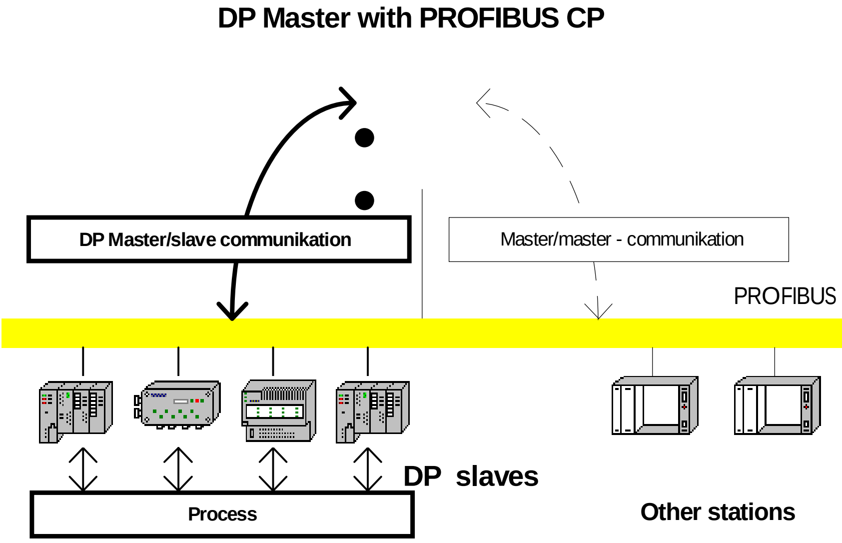


Fig. 1.1: Basic Structure

Definition of DP Slave and DP Master

The PROFIBUS standard DIN 19245 defines two classes of stations:

- passive stations
- active stations

In the distributed I/O system, the I/O devices are passive stations. They are known as **DP slaves**. The DP slaves are controlled by an active master station. This master station is known as the **DP master**.

DP Master Class 1

The DP programming interface allows the use of a PROFIBUS CP in a PG/PC as a DP master class 1. The PC in conjunction with the PROFIBUS CP controls the communication with the distributed I/O devices and other PROFIBUS stations. It implements the central functions of a DP master Class 1 complying with DIN E 19245 Part 3. This means::

- Initialization of the DP system
- Parameter assignment/configuration of the DP slaves
- Cyclic data transfer to the DP slaves
- Monitoring the DP slaves
- Preparing diagnostic information ❑

2 Characteristics of the DP Programming Interface

This chapter provides you with an overview of the characteristics of the DP programming interface of the CP 5412 (A2).

The following sections contain more detailed information about using the various possibilities provided by the interface.

Overview of the Characteristics

- Simple linking of a DP application using the functions of the DP programming interface
- Multi-stage reliability concept
- Data consistency
- Support of single user and multi-user applications
- Support of single board and multiboard applications
- Support of various operating systems and compilers
- Support of slaves belonging to the ET 200 system
- Support of applications created for the “Distributed I/O System ET 200” user interface of the TF-5412/MS-DOS, Windows product.

These points are explained in more detail below.

Simple Linking of DP Applications

The DP programming interface provides you with a range of functions in the form of a library. All the functions have a uniform structure. They allow simple access to the functions of the DP master (Class 1).

The function calls of the DP programming interface are explained in detail in Section 5.

Multi-Stage Reliability Concept

The DP programming interface provides a multi-stage reliability concept to limit the effects of the failure on a communication connection or the DP master.

- ➢ A configurable watchdog for DP slaves ensures that a DP slave that has not been accessed for a longer period of time changes to a safe operating mode.
- ➢ An AUTOCLEAR function can be activated so that if individual DP slaves cannot be accessed, the DP master automatically changes to the CLEAR state.
- ➢ An activity monitoring function can be activated on the DP master to detect inactivity of a DP application and to change the DP slaves controlled by the application to a safe operating mode..

For detailed information about the watchdog, refer to Section 3.7.1. The AUTOCLEAR function is described in detail in Section 3.7.6 and the activity monitoring in Section 5.2.

Data Consistency

When transferring the data between the DP slave and DP application, data consistency is ensured by internal reliability mechanisms.

Single User/Multi-User Operation

In single user operation, only one DP application accesses the DP programming interface. This is the standard application under DOS.

When using operating systems that permit multitasking (for example Windows 3.x, Windows 95 and Windows NT), other separate DP applications can share the DP programming interface. In such applications, the DP programming interface provides mechanisms for coordinating the tasks.

Single Board/Multi-Board Operation

The single board mode means that only **one** PROFIBUS CP is operated in the PG/PC.

In the multiboard mode, **more than one** PROFIBUS CP is operated in the PG/PC. Each of these modules is connected to its own bus. This allows several L2-DP bus systems to be controlled from one computer. Each CP is the DP master on its bus.

For detailed information about the single board and multiboard modes with different operating systems, refer to Chapters 7 and 8.

Operating Systems/ Compilers

The DP programming interface is designed for different operating systems and compilers.

For detailed information about the supported operating systems, compilers, memory model, DP library, include files etc., refer to Chapters 7 and 8.

DP Slaves of the ET 200 System

The DP programming interface of the CP 5412 (A2) supports **all** slaves of the ET 200 system.

Interface calls of the CP 5412 (A1)

The DP programming interface of the CP 5412 (A2) continues to support the DP function calls of the TF-5412/MS-DOS, Windows. product. This product is based on the CP 5412 (A1).

NOTES

3 Basic Principles of Distributed I/Os (DP)

This chapter explains the basic principles of the DP protocol. Understanding the communication structure between the DP master and DP slaves is indispensable for the efficient use of the function calls of the DP programming interface.

This chapter explains the following:

- How the data transfer between the DP master and DP slaves takes place
- How the data structures in the DP master are organized
- The various modes of the DP master
- Which events can be signaled to the DP master
- The various modes of the DP slaves
- Which control frames the DP master sends to the DP slaves
- What you should remember when configuring.

3.1 Communication Between the DP Stations

Polling

Communication between the DP master and the distributed I/O stations takes the form of polling. Polling means that in the productive phase, the DP master sends frames to the DP slaves assigned to it cyclically. Each DP slave is sent its own call frame.

The call frame contains the current output data that the DP slave will apply to its output ports. If a DP slave does not have output ports, an “empty frame” is sent instead.

The reception of a call frame must be acknowledged by the addressed DP slave by returning an acknowledgment frame. The acknowledgment frame contains the current input data applied to the input ports of the DP slave. If a DP slave does not have input ports, an “empty frame” is returned instead.

All the operational DP slaves are addressed in one polling cycle. As soon as the last slave is addressed, a new polling cycle starts. This method ensures that the data are up-to-date. The current input data and diagnostic data of the DP slaves are available to the DP application on the data interface of the DP master. The current output values of the DP application are applied to the output ports of the DP slave.

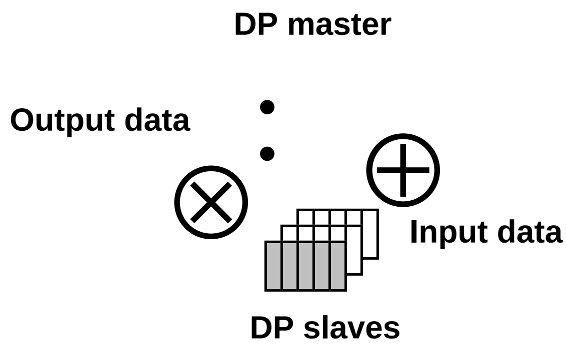


Fig. 3.1: Schematic Representation of the Polling Mode

Diagnostic Messages

In the acknowledgment frame, a DP slave can not only return the current input data but also indicate to the DP master that diagnostic messages are available. Diagnostic messages inform the DP application that special events or errors have occurred on the DP slave, such as a short-circuit, undervoltage, overvoltage, overload, wire break etc.

When it receives the diagnostic message, the DP master reads the diagnostic data using a special call frame and makes this data available to the DP application. The diagnostic data have a uniform structure (see Section 6.2 pp.). This allows the DP application to make a detailed error analysis.

Parameter Assignment/ Configuration

The DP master can only enter a productive data exchange with the DP slaves when it has assigned parameters to them and configured them.

The master assigns parameters and configures the slaves

➢ during the startup phase of the DP master

➢ after a temporary failure of a slave during the productive phase.

The parameter assignment frame sets global operating parameters on the slave (for example the duration of the watchdog).

The configuration frame is sent after the DP slave has had parameters assigned. This contains the current configuration of the DP slave. The configuration contains the number and type of input/output ports. The DP slave compares the received configuration frame with its own values that it recorded during the startup phase. If the values match, the DP slave confirms the configuration and changes to the productive phase.

The parameter assignment and configuration data are specified using the COML DP configuration tool. COML DP creates a database with all relevant parameter assignment and configuration data. This database is loaded on the CP during the startup phase.

3.2 Data Areas on the DP Master

Data Area

Fig. 3.2 shows the data areas of the DP master.

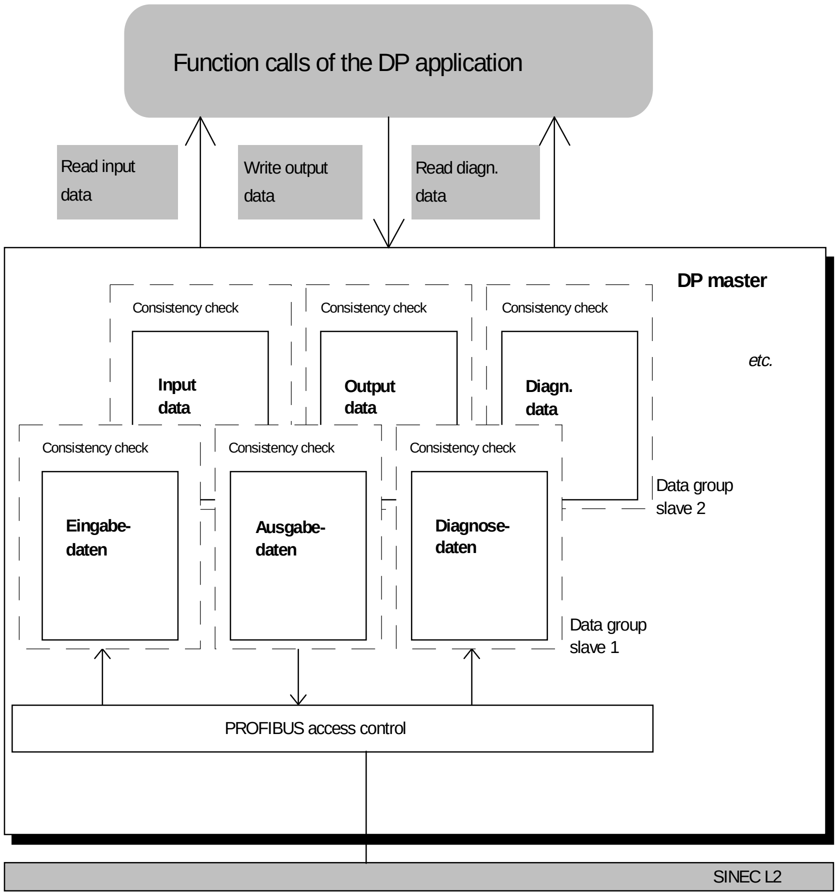


Fig. 3.2: Data Areas of the DP Master

For each configured DP slave, the DP master has three different data areas:

- input data from the DP slave
- output data to the DP slave
- diagnostic data from the DP slave

These areas form a common interface between the CP and the DP application. They are continuously updated during the productive phase. An internal security mechanism ensures the consistency of the data if the DP application and field bus access controller access the data simultaneously. A DP application has access to the data areas using various function calls to the DP programming interface.

Output Data

The data in this area are provided by the DP application. In the productive phase (i.e. after successful parameter assignment/configuration), they are sent to the DP slave cyclically. If no output data exist, an “empty frame” is transmitted instead.

Input Data

During the productive phase, the DP slave sends its input data back to the master in its response frame following each call frame of the DP master. If the DP slave does not have any input ports, it sends an “empty frame” instead. The received response data are entered in the input area of the DP master.

Diagnostic Data

If a DP slave recognizes an error during the initialization or productive phase, it can indicate this to the DP master using a diagnostic request. The received diagnostic data are entered in the diagnostic area of the DP master.

3.3 The Modes of the DP Master

Overview

Communication between the DP master and DP slaves takes place within four modes:

- OFFLINE
- STOP
- CLEAR
- OPERATE

Each of these modes is characterized by defined actions between the DP master and the DP slaves.

Mode	Meaning
OFFLINE	There is no communication whatsoever between the DP master and the DP slaves. This is the initial status of the DP master.
STOP	There is also no communication between the DP master and DP slaves in this mode. In contrast to the OFFLINE mode, a DP diagnostic station (DP master Class 2) can read out diagnostic information of the DP master.
CLEAR	In this mode, the master assigns parameters to and configures all DP slaves entered in the database and activated. Following this, the cyclic data exchange between the DP master and DP slaves begins. In the CLEAR mode, the value 0h is sent to all slaves with process output, i.e. the process output is deactivated. The input data of the slaves are known and can be read out.
OPERATE	The cyclic data transfer to the DP slaves takes place in the OPERATE mode. This is the productive phase. In this mode, the DP slaves are addressed one after the other by the DP master. The call frame contains the current output data and the corresponding response frame contains the current input data.

Setting the Mode

Initially, the DP master is in the OFFLINE mode. To change to the productive phase, in other words to the OPERATE mode, the master must run through the modes above in the following sequence:

OFFLINE -> STOP -> CLEAR -> OPERATE

The DP programming interface provides two ways in which you can change the mode:

➢ After a DP application has logged on, the DP master changes to the OPERATE mode automatically (in other words without any further action by the DP application) and remains in this mode until the DP application is terminated.

➢ After a DP application has logged on, the DP master remains in the OFFLINE mode. The transition to a different mode is triggered by a special function call of the DP application, in other words the DP application itself is responsible for setting the mode.

Which of the two possible methods is used, is specified when the DP application logs on.

Special Case "AUTOCLEAR"

Regardless of the methods explained above, you can also specify during configuration that the DP system changes to a “safe” mode if an error occurs. This function is known as AUTOCLEAR.

☞ **To achieve this reaction, the "Autoclear" option must be set using the configuration tool.**

Effect: If an error occurs on one or more DP slaves during the productive phase, the DP master changes **automatically** to the CLEAR status (the DP system is closed down). In the CLEAR status, the DP master sends data with the value 0h to the DP slaves in the output direction. The DP master no longer exits this status on its own initiative, in other words the user must bring about a change to the OPERATE mode explicitly.

Further Information

The DP application can recognize the current mode of the DP master from the return parameters of certain function calls. For more detailed information about this topic, refer to Section 4.8.3 (sys_state) or Chapter 5 (DP Function Calls).

3.4 The Event Messages of the DP Master

Overview

During the operating phase, unexpected events can occur that are significant for the DP application. In this case, the DP master can inform the DP application of the following events using a return parameter in response to DP function calls:

Event Message	Meaning
Autoclear	Automatic closing down of the DP system to the CLEAR mode, when errors occur in communication with DP slaves. Requirement: The AUTOCLEAR function must be configured in COML DP.
Timeout	The watchdog time of the DP application has expired. Cause: The DP application has not made a DP function call during the time preset by the application. Requirement: The DP application must have logged on and transferred a watchdog time to the DP master. The required function calls are described in Chapter 5.
Access by a DP master Class 2	The DP master Class 2 is a special DP diagnostic station that can perform detailed on-line diagnostics of the DP master Class 1 and DP slaves. This event message signals that a DP diagnostic station is taking part in the bus traffic and is currently accessing internal diagnostic lists of the DP master. Note: With the current DP firmware, no special reaction to the event message is normally required of the DP application since the CP normally handles data exchange with the diagnostic station automatically. The event message is a “place holder” intended for future expanded diagnostic DP functions in which the DP application will have the option of coordinating certain diagnostic sequences with the DP diagnostic station.

Further Information

With function calls from a DP application, the DP master enters the event messages in a special return parameter. For detailed information about the constants occurring in event messages, refer to Section 4.8.4 and the function calls in Chapter 5.

3.5 The Operating Status of the DP Slaves

Overview

During the operational phase, the DP master evaluates the acknowledgment frames received from the DP slaves. Based on these frames, the DP master can recognize the current operating status of the DP slaves. Using some of the DP function calls, the DP application can query these values.

The following operating statuses of a DP slave can be signaled to the DP application:

Operating Statuses

- ➢ The DP slave is in the data transfer phase.
- ➢ The DP slave is in the data transfer phase, and diagnostic data exist.
- ➢ The DP slave is not in the data transfer phase (CP starting up).
- ➢ The DP slave is not in the data transfer phase.
- ➢ The DP slave is not in the data transfer phase, and diagnostic data exist.
- ➢ The DP slave is not activated.

Further Information

With certain function calls from the DP application, the DP master enters the operating status of a slave in a return parameter. For detailed information of the constants used for this, refer to Section 4.8.2 and to the function calls in Chapter 5.

3.6 Control Frames to One or More Slaves

Overview

During the configuration, a slave can be assigned a group identification, in other words it is possible to group several slaves together (up to 8 different groups can be formed). In the productive phase, individual groups can then be addressed using control frames (known as global control frames).

A control frame is a frame that the master sends to one slave, a group, several groups or to all slaves. These frames are not acknowledged by the slaves.

Control frames are used to transfer control commands (known as global control commands) to the selected slaves to allow synchronization. A control command contains three components:

- ➢ Identifier whether one or more DP slaves are being addressed
- ➢ Identification of the slave group
- ➢ Control command

Creating Groups

You specify which slaves belong to a group when creating the database with the COML DP configuration tool. During this phase, each DP slave can be assigned a group number. The DP slave is informed of this group number during the parameter assignment phase. You can specify a maximum of eight groups.

Control Commands

The following control commands can be sent to DP slaves:

- ➢ **FREEZE** The signal state of the inputs is read in and frozen.
- ➢ **UNFREEZE** This cancels the freezing of the inputs.
- ➢ **SYNC** Output is frozen.
- ➢ **UNSYNC** The UNSYNC command cancels the SYNC command.
- ➢ **CLEAR** All the outputs are reset.

3.6.1 Synchronization

Control frames can be sent to the slaves cyclically or acyclically.

Cyclic Transmission of Control Frames

Fig. 3.3 illustrates cyclic transmission. During configuration with COML DP, you can specify whether a slave will be operated in the SYNC mode, in the FREEZE mode, in the SYNC and FREEZE mode or in neither of these modes.

The DP master automatically takes into account that certain slaves must be operated in a certain mode. Once data transfer has been completed with all slaves, the master waits for a set time (min slave interval, see Section 3.7.4), to allow all slaves operating in the SYNC mode, to process the previously transferred output data.

At the end of this interval, the DP master sends a control frame to all slaves operating in the SYNC and/or FREEZE mode. The effect of this is that all slaves operating in the SYNC mode switch the previously transferred data to their process output cyclically. All slaves operating in the FREEZE mode read in the input data of the process when they receive this frame.

After sending the frame, the DP master again waits for a set time before it begins a new data transfer cycle. This gives the slaves operating in the FREEZE mode the opportunity to prepare the process input they have just read in synchronously for the later response frame to the master (in the next data transfer cycle).

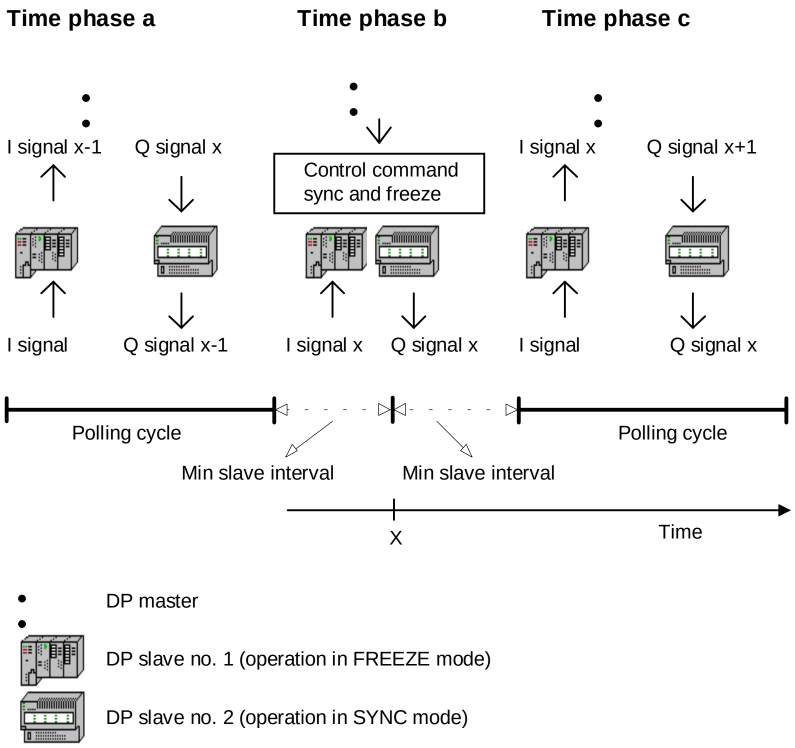


Fig. 3.3: Cyclic Transmission Based on the Example of Two DP Slaves

Cyclic Transmission (Example)

The figure illustrates three time phases.

Time Phase a

During the polling cycle, the DP master requests the input data from slave number 1. The slave replies with the input data that it read in when it received the last control frame. The DP master then sends new output data to slave number 2, but this is not output by the slave.

Time Phase b

At time x, the DP master sends a control frame to the DP slaves. On receiving the frame, DP slave number 1 reads in the input signal x. At the same time, slave number 2 transfers the output data that it received from the DP master during the previous polling cycle to the process.

Time Phase c

During the polling cycle, slave number 1 replies with the input value read in at time x. The DP master sends new output data to slave number 2.

Acyclic Transmission

The DP programming interface also provides the option of sending control frames to certain slaves acyclically, in other words not within the framework of the polling cycle or configured min slave interval.

To allow this, the DP application has a special function call with which the DP master is instructed to send a control frame once.

Further Information

For detailed information about the constants for individual control commands, refer to Section 4.8.5, and for the function call for acyclic transmission of control frames refer to Section 5.17.

Please remember that not all slaves support operation in the SYNC or FREEZE mode. Refer to the manufacturer's instructions.

3.7 Notes on Configuration

General

To be able to communicate with the DP slaves, the DP master requires a database. The database contains all the operating and configuration data required for a DP system.

You create the configuration using COM PROFIBUS. This generates the database of the CP.

For detailed information about configuring, refer to the volume “SIMATIC ET 200 Distributed I/O System”.

Loading the Database

The database and firmware are loaded on the CP 5412 (A2) during the system startup phase. Once the firmware has been loaded, the CP is an active station with the role of DP master complying with the PROFIBUS standard DIN 19245 Part 1.

There is no data exchange between the CP and DP slaves until a DP application has logged on.

3.7.1 Watchdog

The watchdog of a DP slave can be activated or deactivated by the DP master during the parameter assignment phase (depending on the information in the configured database). If the watchdog of a DP slave is activated, the DP master must communicate with the DP slave within a set time. If there is no communication during this time, the slave switches its outputs to a safe state and no longer takes part in the data transfer with the master, since the slave assumes that a serious error (for example wire break, DP master failure) has occurred. The master must then re-assign parameters and re-configure the slave. The exchange of productive data is only possible again when this has been completed.

The description of the DP slaves explains which “safe” value is applied to the outputs.

3.7.2 Data Control Time

The data control time is fixed. Once the data control time has expired, the DP master checks whether all the slaves configured in the database are taking part in data exchange. The slaves that are configured but that are also de-activated when the data control time expires are not counted (see also Section 3.7.5). If one of the activated slaves is not taking part in the data exchange and if the AUTOCLEAR function is set, the DP master automatically changes to the CLEAR mode.

Once this time has expired, the DP master also informs the DP slaves of its operating status.

3.7.3 Poll Timeout

The poll timeout time can be set by a function call of the DP programming interface. The time is used to monitor the communication with a DP master of Class 2 (DP diagnostic master). The DP diagnostic master can request certain information from the DP master Class 1 (CP 5412 (A2)) during operation (for example the mode of the DP master Class 1). If the job can be executed, the DP master Class 1 prepares the data to be fetched. If the DP diagnostic master does not fetch this data within the poll timeout time, the data is cleared.

3.7.4 Min Slave Interval

You configure the min slave interval using COML DP and it can be used in two ways.

➢ If cyclic control frames are configured, the time is used as shown in Fig. 3.3.

➢ If no cyclic control frames are configured, the time is used as shown in Fig. 3.4.

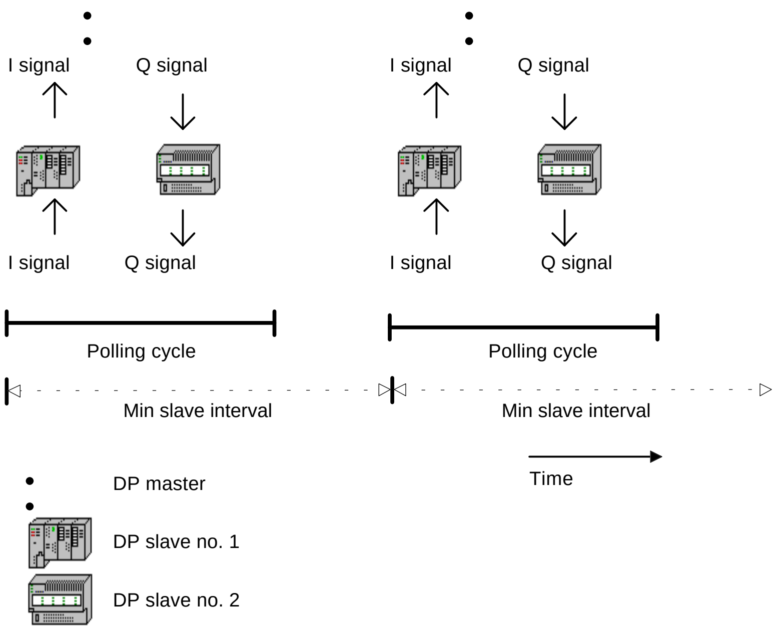


Fig. 3.4: Min Slave Interval - Without Cyclic Control Frame

3.7.5 Deactivating a DP Slave

A DP slave can be activated or deactivated using a function call of the DP programming interface. DP slaves that are deactivated are ignored in the polling cycle.

3.7.6 AUTOCLEAR

This function allows the DP master to monitor the DP system automatically. If you have selected this function, the master checks whether all the slaves that have not been deactivated (Section 3.7.5) are taking part in the data exchange when the data control time (Section 3.7.2) expires. If this is not the case, the DP master changes to the CLEAR mode.

3.7.7 Configuration Data

The database must contain the configuration data for every DP slave. The configuration specifies the number and type (input/output/analog/digital) and the consistency (byte/word/area) of the data areas. This configuration data must match the actual configuration of the DP slave.

The DP master sends this data to the DP slave in a configuration frame. The DP slave then compares the received values with its actual configuration. Productive data can only be exchanged between the DP master and DP slave when this information matches.

If the values do not match, the DP slave signals a configuration error. ❑

NOTES

4 Structure of the DP Programming Interface

This chapter provides you with an overview of the function calls provided by the DP programming interface. The function calls are divided into several groups according to their meaning.

The chapter also describes the basic structure of a DP application that can be divided into several functional sections.

The appropriate function calls are dealt with in these sections. From the description you can see which function calls are mandatory and which function calls can be used as options.

This chapter also explains the structure and elements of the data structures transferred to the functions as call parameters.

The pre-defined constants that are assigned to the different elements of the data structures are also described. A distinction is made between call and return parameters.

The return parameters of the function calls are listed and briefly described in the table.

4.1 Overview of the DP Call Functions

Grouping of the DP Functions

The DP call functions can be divided into the following groups: initialization functions, database functions, data transfer functions, control functions and close functions.

Initialization Functions

dpn_init()	Log on a DP application at the DP programming interface
dpn_wd()	Activate monitoring of the DP application

Database Functions

dpn_read_bus_par()	Read out the bus parameters from the database
dpn_load_bus_par()	Modify DP-specific parts of the bus parameters
dpn_read_slv_par()	Read slave parameters
dpn_set_slv_state()	Activate/deactivate a slave of the database
dpn_read_cfg()	Read out the DP configuration

Diagnostic Functions

dpn_slv_diag()	Request diagnostic data of a slave
dpn_read_sys_info()	Read out system information

Data transfer Functions

dpn_out_slv()	Send output data to a single slave
dpn_out_slv_m()	Send output data to several slaves
dpn_in_slv()	Read input data from a single slave
dpn_in_slv_m()	Read input data from several slaves
dpn_read_slv()	Read local output data of a single slave

Control Functions

dpn_set_mode()	Set the DP mode
dpn_get_mode()	Read out the current DP mode
dpn_global_ctrl()	Send control commands to a slave group

Close Functions

dpn_reset()	Log off a DP application at the DP programming interface

4.2 General Call for the DP Functions

```
Return value = dpn_..(struct dpn_interface far *  ptr);
or
Return value = dpn_..(struct dpn_interface_m far *  ptr);
```

As a transfer parameter, each function expects a pointer to the **dpn_interface** structure. The elements of this structure are described in Section 4.5.

Exception

The **dpn_interface _m** structure is used for the two call functions dpn_in_slv_m() and dpn_out_slv_m(). This structure is also described in Section 4.5.

Both structures are defined in the dpn_user.h file.

4.3 Evaluating a Function Call

Two Evaluation Methods

A DP application can use one of the two following methods to check whether a function call was processed:

- Evaluation of the return value of the DP function
- Evaluation of the error_code structure element of the structures dpn_interface or dpn_interface_m

The content of the error_code structure element is identical to the return value of the DP function.

4.3.1 Evaluating the Return Value

Checking the Return Value of the Function

Using the return value of the function call (type: unsigned short), the calling DP application can check whether the function was processed.

Return value of the function call	Meaning
DPN_NO_ERROR	The function was processed. The return parameters of the dpn_interface structure (dpn_interface_m) are valid.
Not DPN_NO_ERROR	The processing of the function was aborted due to an error. The return parameters of the dpn_interface structure (dpn_interface_m) are not valid. The return parameter of the function contains a detailed error ID.

Example

Checking the return value using the dpn_get_mode() function as an example.

```
#include "dpn_user.h"

struct dpn_interface myDpnInterface;
/* get actual mode of DP-System */
unsigned char GetActualMode   (unsigned char myBoard,
                               unsigned char  myAccess)
{  unsigned short int result;
   unsigned char mode;
       myDpnInterface.reference.board_select = myBoard;
       myDpnInterface.reference.access = myAccess;
       result = dpn_get_mode (&myDpnInterface);
       if (result == DPN_NO_ERROR) /* result is valid */
       {
         mode = myDpnInterface.sys_state;
       }
       else
       {
         mode = 0xff; /* sign for invalid */
         switch (result)
         {
            /* check error code */
         }
       }
       return (mode);
}
```

4.3.2 Evaluating the Structure Element "error_code"

Checking the Structure Element "error_code"

An alternative to the return parameters of the function call (see above), a DP application can also evaluate the error_code structure element to determine whether the function was processed or not. The error_code structure element in the dpn_interface(_m) structure is identical to the return parameter of the function.

Structure element error_code	Meaning
DPN_NO_ERROR	The function was processed. The remaining return parameters of the dpn_interface (dpn_interface_m) structure are valid.
Not DPN_NO_ERROR	The processing of the function was aborted due to an error. The remaining return parameters of the parameters of the dpn_interface (dpn_interface_m) structure are not valid. The error_code structure element contains a detailed error ID.

Example

Checking the error_code structure element based on the example of the dpn_get_mode() function.

```
#include "dpn_user.h"

struct dpn_interface myDpnInterface;
/* get actual mode of DP-System */
unsigned char GetActualMode(unsigned char myBoard,
                              unsigned char  myAccess)
{  unsigned char mode;
       myDpnInterface.reference.board_select = myBoard;
       myDpnInterface.reference.access = myAccess;
       dpn_get_mode (&myDpnInterface);
       if (myDpnInterface.error_code == DPN_NO_ERROR)
       /* result is valid */
       {
         mode = myDpnInterface.sys_state;
       }
       else
       {
         mode = 0xff; /* sign for invalid */
         switch (myDpnInterface.error_code)
         {
            /* check error code */
         }
       }
       return (mode);
}
```

4.4 Overview of the Error IDs

List of Error IDs

The table contains a list of the error IDs of the DP functions. For more information about the term “central application” used in the table, refer to Sections 4.8.1 and 5.1.3.

Error ID	Meaning
DPN_NO_ERROR	No error processing the function call.
DPN_ACCESS_ERROR	The calling DP application does not have right of access to the function or to a slave.
DPN_APPL_LIMIT_ERROR	The maximum permitted number of DP applications in the multi-user mode has been exceeded.
DPN_CENTRAL_ERROR	The function can only be called by a central DP application.
DPN_CLOSE_ERROR	Error logging off a DP application.
DPN_LENGTH_ERROR	Structure element length of the dpn_interface structure is outside the permitted range of values. The data length does not match the configured value.
DPN_MEM_BOARD_ERROR	Not enough free memory on the CP.
DPN_MEM_HOST_ERROR	Not enough free memory on the host.
DPN_MODE_ERROR	The function call cannot be processed in the current mode or a status was skipped when changing the mode.
DPN_NO_DBASE_ERROR	No or incorrect entries in the DP database.
DPN_OPEN_ERROR	Error logging on a DP application (e.g. driver not loaded, CP not plugged in).
DPN_RECEIVE_ERROR	Error transferring an acknowledgment from the CP to the DP application.
DPN_REFERENCE_ERROR	The reference structure element of the dpn_interface structure is not valid.
DPN_REFERENCE_DIFF_ ERROR	Error in the multi-user mode. The reference structure element with the dpn_init() call does not match the previous dpn_init() of other DP applications that have logged on at the same CP.
DPN_SEND_ERROR	Error transferring a function call to the CP.
DPN_SLV_STATE_ERROR	The slv_state structure element of the dpn_interface structure is not valid.
DPN_STAT_NR_ERROR	The stat_nr structure element of the dpn_interface structure is not valid (e.g. slave does not exist in the database).
DPN_USER_DATA_ERROR	One or more elements of user_data array of the dpn_interface structure are invalid.
DPN_WRONG_BOARD_ERROR	The reference.board structure element of the dpn_interface structure is not valid.
DPN_SYS_STATE_ERROR	The sys_state structure element of the dpn_interface structure is not valid.
DPN_GLB_CTRL_ERROR	Invalid range of values for control command when calling the dpn_global_crtl() function.
DPN_BOARD_ERROR	CP error.
DPN_WD_EXPIRED	The job could not be executed because the watchdog of the DP application detected a timeout.

4.5 Transfer Structures

A pointer to the dpn_interface or dpn_interface_m structure is transferred to a DP function as a parameter. The structure contains the call and return elements of the DP function.

Both structures are contained in the dpn_user.h include file.

The dpn_interface_m structure is used when the call involves data transfer to several DP slaves. It contains a separate substructure dpn_interface_s for each slave.

dpn_interface Structure

```
struct dpn_interface
{
struct REFERENCE  reference;      /* Reference of applic.  */
unsigned char     stat_nr;        /* Station number        */
unsigned char     length;         /* Length                */
unsigned short    error_code;     /* Error identifier      */
unsigned char     slv_state;      /* Status of the DP slave */
unsigned char     sys_state;      /* Status of the DP master*/
unsigned char     sys_event;      /* Event messages        */
unsigned char     user_data[255]; /* Data field            */
};

struct REFERENCE
{
unsigned char     board_select;
unsigned char     access;
};
```

Structures dpn_interface_m, dpn_interface_s

```
struct dpn_interface_s
{
struct REFERENCE      reference;
unsigned char         stat_nr;
unsigned char         length;
unsigned short int    error_code;
unsigned char         slv_state;
unsigned char         sys_state;
unsigned char         sys_event;
unsigned char         user_data[DPN_SINGLE_SIZE];
};

struct dpn_interface_m
{
struct dpn_interface_s dpn_if_single[DPN_MULTIPLE_SIZE];
};
```

4.6 Description of the Structure Elements

reference
The reference structure element contains the identification of the DP application. The identification is assigned when the dpn_init() function is called. Based on this entry, the DP library can assign the function call to the selected CP.

stat_nr
L2 address of the selected DP slave.

length
Number of valid bytes in the user_data[ ] array.

error_code
This structure element is **identical** to the return parameter of the function.
This signals whether the function call could be processed or not.

Note:
A DP application can either check the return value of the function or the error_code structure element.

Remember that the remaining structure elements are only valid when the function return parameter or the error_code structure element do not contain an error ID.

slv_state
This structure element indicates the status of the addressed DP slave.

sys_state
This structure element indicates the current mode of the DP master.

sys_event
If an error occurs on the DP master (see also Section 3.4), it enters an appropriate ID at this point to identify the event.

user_data[ ]
The user_data[ ] array contains the data specific to the job.

4.7 Assignment of the Parameters to the DP Functions

Description

The following matrix illustrates the assignment of the parameters of the dpn_interface and dpn_interface_m structures as described in Section 4.6 to the function calls of the DP library.

A "C" in the parameter matrix means that the parameter is used as a call value.

An "R" in the parameter matrix indicates the return parameters of the function calls.

Function Calls	reference	stat_nr	length	error_code	slv_state	sys_state	sys_event	user_data
dpn_init()	C/R	-	C	R	-	-	R	C
dpn_wd()	C	-	C	R	-	R	R	C
dpn_read_bus_par()	C	-	C/R	R	-	R	R	R
dpn_load_bus_par()	C	-	C	R	-	R	R	C
dpn_read_slv_par()	C	C	C/R	R	C/R	R	R	R
dpn_set_slv_state()	C	C	C	R	R	R	R	C
dpn_read_cfg()	C	-	C/R	R	-	R	R	R
dpn_slv_diag()	C	C	C/R	R	R	R	R	R
dpn_read_sys_info()	C	-	C/R	R	-	R	R	R
dpn_out_slv()	C	C	C	R	R	R	R	C
dpn_out_slv_m()	C	C	C	R	R	R	R	C
dpn_in_slv()	C	C	C/R	R	R	R	R	R
dpn_in_slv_m()	C	C	R	R	R	R	R	R
dpn_read_slv()	C	C	C/R	R	R	R	R	R
dpn_set_mode()	C	-	-	R	-	C/R	R	-
dpn_get_mode()	C	-	-	R	-	R	R	-
dpn_global_ctrl()	C	C	C	R	C	R	R	C
dpn_reset	C	-	-	R	-	-	-	-

4.8 Constants

The following tables contain the definitions of constants that are entered as call or return parameters in the dpn_interface(_m) structure. These constants are referred to in Chapter 5 in the description of the function calls.

The constants are defined in the dpn_user.h include file.

4.8.1 "reference" Structure Element

Type of Application

The DP programming interface distinguishes between two types of DP applications - central and not central.

Certain function calls are reserved for the central DP application. Of the maximum four DP applications that can log on at the DP programming interface, a maximum of one can be logged on as the central DP application per CP. For further information about this topic, refer to Section 5.1.3.

reference.access

With the dpn_init() function call, these constants indicate the type and environment of the DP application. For a detailed description of the dpn_init() function call, refer to Chapter 5.

Possible Entries	Meaning
DPN_ROLE_CENTRAL	The DP application that is logging on is a central application.
DPN_ROLE_NOT_CENTRAL	The DP application that is logging on is not a central application.
DPN_SYS_CENTRAL	One of the DP applications is a central application.
DPN_SYS_NOT_CENTRAL	None of the DP applications is a central application.

Right of Access to DP Slaves

With the dpn_init() function call, these constants specify the access rights of a DP application to a DP slave and are entered in the user_data[ ] array.

Possible Entries	Meaning
DPN_SLV_NO_ACCESS	No access requested
DPN_SLV_READ	Only read input data
DPN_SLV_WRITE_READ	Send output data/read input data

4.8.2 "slv_state" Structure Element

Statuses of a DP Slave

These constants indicate the current operating status of a DP slave during the communication with the DP master.

Possible Entries	Meaning
DPN_SLV_STAT_OFFLINE	The DP slave is not in the data transfer phase (CP starting up).
DPN_SLV_STAT_NOT_ACTIVE	The DP slave is not activated in the local database.
DPN_SLV_STAT_READY	The DP slave is in the data transfer phase.
DPN_SLV_STAT_READY_DIAG	The DP slave is in the data transfer phase and diagnostic data exist.
DPN_SLV_STAT_NOT_READY	The DP slave is not in the data transfer phase.
DPN_SLV_STAT_NOT_READY_DIAG	The DP slave is not in the data transfer phase and diagnostic data exist.

4.8.3 "sys_state" Structure Element

Mode of the DP Master

These constants indicate the current mode of the DP master.

Possible Entries	Meaning
DPN_SYS_OFFLINE	There is no DP communication.
DPN_SYS_STOP	Only communication with DP master Class 2 is possible. There is also no communication with DP slaves.
DPN_SYS_CLEAR	The DP slaves are in the parameter assignment/configuration phase. In the data transfer phase that follows, bytes with logical 0 are sent to the DP slaves in the output direction.
DPN_SYS_OPERATE	Productive data exchange with the DP slaves.

4.8.4 "sys_event" Structure Element

Event Messages

These constants identify event messages that can be indicated in the sys_event structure element with the various DP functions. Note that the event messages can occur singly or in combinations. If no event has occurred, sys_event contains the value 0. If sys_event contains a value other than 0, the number and type of events can be determined using “AND operations” on the various constants.

Possible Entries	Meaning
MST_CLS_TWO_ACCESS	Event message access by a DP master Class 2 (diagnostic master).
AUTOCLEAR	Event message AUTOCLEAR
WATCHDOG	Event message TIMEOUT

Example

```
Simple example:
struct dpn_interface myDpnInterface;
// Here complete the parameters and execute the
// call
       ::
// Check whether Autoclear has occurred
if ((myDpnInterface.sys_event & AUTOCLEAR) != 0)
{
 /* Event AUTOCLEAR */
}
```

4.8.5 Global Control Commands

Using the dpn_global_ctrl() function call, you can send various commands to the DP slaves. The constants listed below are entered in the slv_state structure element.

Possible Entries	Meaning
DPN_CLEAR	CLEAR command
DPN_UNFREEZE	UNFREEZE command
DPN_FREEZE	FREEZE command
DPN_UNSYNC	UNSYNC command
DPN_SYNC	SYNC command

4.8.6 Activating/Deactivating a Slave

These constants are relevant for the dpn_set_slv_state() function call and are entered in the user_data[0] structure element.

Possible Entries	Meaning
DPN_SLV_ACTIVATE	The slave is activated in the local database. Following this, the DP master attempts to assign parameters to the slave, to configure the slave and to execute data transfer. (Condition: The master is in the CLEAR or OPERATE mode.)
DPN_SLV_DEACTIVATE	The slave is deactivated in the local database, i.e. there is no data exchange with the slave.

4.8.7 Slave Parameters

Using these constants, different parts of the slave parameters (structure complying with DIN E 19245 Part 3) can be selected in the dpn_read_slv_par() function call. The constant is entered in the slv_state structure element.

Possible Entries	Meaning
DPN_SLV_PARA_TYP	Selection of Sl-Flag, Slave-type and Octet-String
DPN_SLV_PARA_PRM_DATA	Selection of the parameter assignment data
DPN_SLV_PARA_CFG_DATA	Selection of the configuration data
DPN_SLV_PARA_ADD_TAB	Selection of the Add-Tab list
DPN_SLV_PARA_USER_DATA	Selection of the slave user data

4.8.8 DP Slave Types

These constants indicate the different types of DP slaves and are returned with the dpn_read_cfg() function call in the user_data[ ] field.

Possible Entries	Meaning
DPN_CFG_NO_SLV	No DP slave
DPN_CFG_NORM	Standard DP slave
DPN_CFG_ET200_U	Non-standard slave: ET 200 U
DPN_CFG_ET200K_B	Non-standard slave: ET 200 K/B
DPN_CFG_ET200_SPM	Non-standard slave: General SPM station

4.9 Structure of a DP Application

Overview

Fig. 4.1 illustrates the basic structure of a DP application. It is divided into the three areas: Initialization, productive phase and close phase.

Certain DP function calls are assigned to each of these areas. The function calls that are mandatory for a DP application are shown on a gray background.

In the left-hand column, the figure shows the typical functions of a DP application. The right-hand column contains the additional functions available for extended diagnostics, modifying parameters etc.

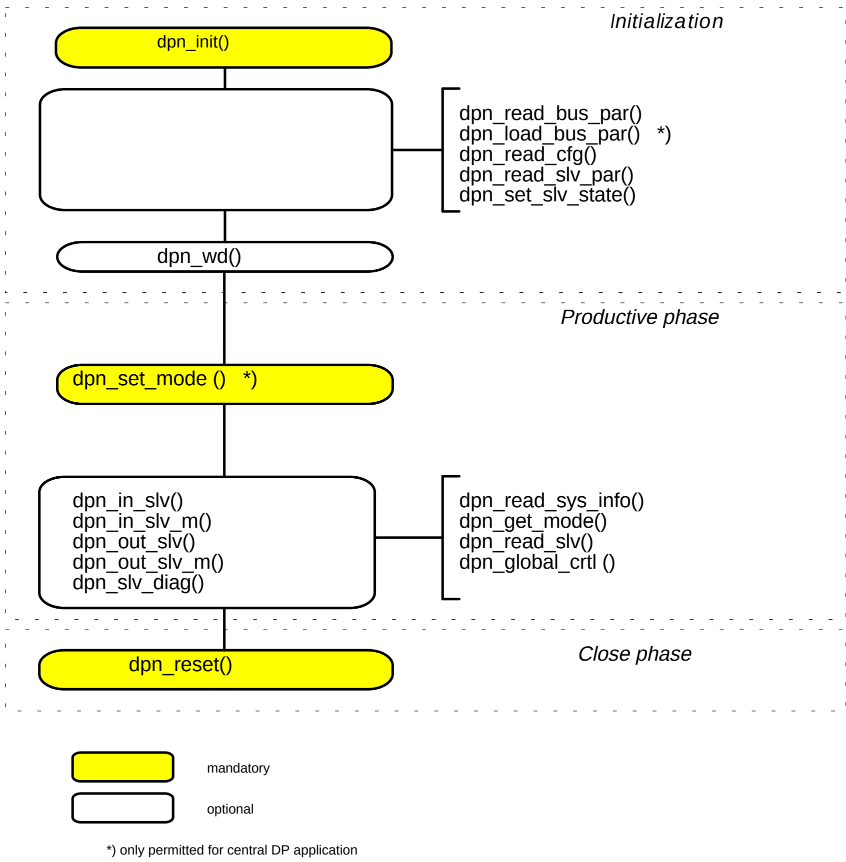


Fig. 4.1: Structure of a DP Application

Functions of the Initialization Phase

Function	Status	Comment
dpn_init()	mandatory	Must be called as the first DP function.
dpn_read_cfg()	optional	Read out the complete configuration.
dpn_read_bus_par()	optional	Read/check the bus parameters.
dpn_load_bus_par()	optional or not allowed	Change the DP timers of the bus parameters. This must only be called by a central DP application.
dpn_read_slv_par()	optional	Read out the slave parameters. From the slave parameters, the configuration and status (activated/not activated) of the DP slaves can be obtained.
dpn_set_slv_state()	optional	Activate/deactivate slave.
dpn_wd()	optional	CP monitors the activity of the DP application.

Functions of the Productive Phase

Function	Status	Comment
dpn_set_mode()	mandatory or not allowed	Must be called by the central DP application at the start of the operating phase to set a new mode on the DP master.
dpn_get_mode()	optional	Queries the current mode.
dpn_out_slv()	optional	Transfer output data to a DP slave.
dpn_out_slv_m()	optional	Transfer output data to several DP slaves.
dpn_in_slv()	optional	Read the input data of a DP slave.
dpn_in_slv_m()	optional	Read the input data of several DP slaves.
dpn_read_slv()	optional	Read the output data of a DP slave.
dpn_slv_diag()	optional	Read the diagnostics of a DP slave.
dpn_read_sys_info()	optional	Overview of the total status.
dpn_global_ctrl()	optional or not allowed	Control command to several or all DP slaves.

Special Case dpn_set_mode

This function is only mandatory in a DP system with a central DP application. In this case, the DP application must change the master from the OFFLINE to the OPERATE mode using the dpn_set_mode() function.

On a DP system without a central DP application, the DP master changes from the OFFLINE mode to the OPERATE mode automatically after the first application logs on.

The principles of the central DP application are described in detail in Section 5.1.

Functions of the Close Phase

Function	Status	Comment
dpn_reset()	mandatory	The application logs off and terminates DP communication with the slaves assigned to the application. The operating system resources are returned. ❑

NOTES

5 Description of the DP Functions

This chapter contains a detailed description of all the functions of the DP programming interface.

From the description, you can see which elements of the dpn_interface or dpn_interface_m structure must be completed before the function is called.

The chapter also explains the structure elements that are updated or modified by the DP function as return parameters. These elements can be evaluated in the DP application providing the return parameter (or the error_code structure element) signals correct execution.

Overview of the functions:

- dpn_init() Section 5.1
- dpn_wd() Section 5.2
- dpn_read_bus_par() Section 5.3
- dpn_load_bus_par() Section 5.4
- dpn_read_slv_par() Section 5.5
- dpn_set_slv_state() Section 5.6
- dpn_read_cfg() Section 5.7
- dpn_slv_diag() Section 5.8
- dpn_read_sys_info() Section 5.9
- dpn_out_slv() Section 5.10
- dpn_out_slv_m() Section 5.11
- dpn_read_slv() Section 5.12
- dpn_in_slv() Section 5.13
- dpn_in_slv_m() Section 5.14
- dpn_set_mode() Section 5.15
- dpn_get_mode() Section 5.16
- dpn_global_ctrl() Section 5.17
- dpn_reset() Section 5.18

5.1 How a DP Application Logs On

Call Function **unsigned short int dpn_init (struct dpn_interface far * ptr)**

Description Using this function, a DP application **must** log on at the CP. **The function must be called before all other DP functions**. A far pointer to the dpn_interface structure is transferred as the call parameter. If the call is successful, a **handle** is returned in the reference structure element. This handle is required by the DP firmware to identify the DP application. The handle must be entered as a call parameter for all further function calls to the CP.

☞ **After the DP master has been changed to the OPERATE state, the DP slaves may still require a certain amount of time until they are ready for operation. Before writing output data for a DP slave, you should therefore check the operating mode of the DP slaves, for example using dpn_read_sys_info until the value DPN_SLV_STAT_READY or DPN_SLV_STAT_READY_DIAG is returned for the DP slave.**

5.1.1 Call Parameters

ptr -> **reference**	**Reference of the DP application**
ptr -> stat_nr	irrelevant
ptr -> **length**	**Size of the user_data[ ] array**
ptr -> error_code	irrelevant
ptr -> slv_state	irrelevant
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> **user_data[ ]**	**Required type of access to DP slaves**

reference The parameters relevant to the DP application are entered in the reference structure element.

These are as follows:

- ➢ Selection of a module
- ➢ Setting the DP application environment
- ➢ Setting the type of application

You will find a detailed description of these parameters in Section 5.1.3.

length Here, the number of relevant bytes in the user_data[ ] array must be entered; range of values: 0 to 126.

user_data[ ] The DP application must specify the required access rights to the individual DP slaves in the user_data[ ] array. The entry is made in a separate byte for each slave. The index in the array corresponds to the L2 address of the DP slave.

Right to access slave 0	user_data[0]
Right to access slave 1	user_data[1]
Right to access slave 2	user_data[2]
Right to access slave 3	user_data[3]
Right to access slave 4	user_data[4]
Right to access slave 5	user_data[5]
:	:
Right to access slave 125	user_data[125]

Fig. 5.1: Access Rights to the DP Slaves

The required access rights can be as follows:

- ➢ Write output data/read input data
- ➢ Read input data
- ➢ No access required

When it specifies the access rights, the DP application can use the following constants (dpn_user.h include file):

Constant	Required Access Right
DPN_SLV_WRITE_READ	Write output data/ read input data
DPN_SLV_READ	Read input data
DPN_SLV_NO_ACCESS	No access required

Checking the Entries

The validity of the entries is checked as follows:

- **Data output** to a particular DP slave must only be made by **one single** DP application within a multitasking operating system. This application must enter the access right DPN_SLV_WRITE_READ when it logs on. Attempts by other applications to send data to this slave are rejected.

  This access protection prevents undefined dangerous situations arising in the plant that could otherwise occur if two DP applications can send output data to the same DP slave without being coordinated (example: DP application A closes a valve, DP application B opens the valve).

- When the application logs on, the program checks whether the option write output/read input data has already been set by a different DP application. If this is the case, the dpn_init() function is acknowledged negatively.

  Any DP application always has the right to read a slave's data (i.e. image of the input data, image of the output data and image of the diagnostic data).

5.1.2 Return Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> stat_nr	irrelevant
ptr -> length	unchanged
ptr -> **error_code**	**Error identifier**
ptr -> slv_state	irrelevant
ptr -> sys_state	irrelevant
ptr -> **sys_event**	**Event messages**
ptr -> user_data[ ]	unchanged

reference The reference structure element (parameters reference.board_select and reference.access) contains a handle that must be used in all further DP function calls to this CP.

error_code The error_code structure element is identical to the return value of the function call. The remaining return parameters of the dpn_interface structure are only valid if no error occurred (DPN_NO_ERROR).

The possible error identifiers are described in Section 4.4.

sys_event The structure element sys_event identifies event messages during DP communication.

The possible event messages are described in Sections 3.4 and 4.8.4.

5.1.3 Explanation of the "reference" Structure Element

reference.board_ select

Several CPs can be installed in one PG/PC. A DP application must log on at every CP with which it wants to communicate using a separate dpn_init() call.

The number of the CP (1,2...) must be transferred in the reference.board_select structure element.

reference.access

The reference.access structure element contains the "DP application type" and "DP application environment" identifiers. The meaning of these identifiers is explained below.

DP Application Environment

This identifier determines the start-up response and the permitted DP functions when several independent DP applications of a multitasking operating system use the DP functions of the **same** CP.

There are two DP application environments:

➢ Of the DP applications that access the **same** CP, **one** is logged on as the central DP application. This application has full access to all DP functions.

➢ All the DP functions accessing the **same** CP are of equal rank. There is **no** central DP application.

DP Application Type

This identifier specifies the type of DP application.

The following two values are possible:

- ➢ The DP application is a central application.
- ➢ The DP application is not a central application.

A Central DP Application Exists

In this case, the DP communication is controlled **centrally** by a selected DP application. This DP application has access to **all** DP functions. A change in the mode of the DP master is only possible using a dpn_set_mode() call from the central DP application. The central DP application can change the mode at any time using a dpn_set_mode() call. If the central DP application sends a dpn_reset() call, communication is terminated with all DP slaves.

The following function calls are reserved for the central DP application:

- ➢ dpn_set_mode() sets the mode
- ➢ dpn_global_ctrl() sends a control frame to DP slaves
- ➢ dpn_load_bus_par() modifies bus parameters

Example

To control a DP system, two Windows DP applications (tasks) are created. Both tasks access the same CP. One of the tasks is identified as the central task. The central task implements the control of the DP system (access to input and output ports of the DP slaves). The other task is responsible for visualization of the DP system (for example displaying whether valves are open or closed). In this case, the relevant functions such as setting output ports starting and stopping the DP system etc. are only executed by the central task.

No Central DP Application Exists

In this case, there is no special task to control the way in which DP communication is handled. All the DP applications have the same rank. Communication with DP slaves is started when the first DP application logs on successfully using the dpn_init() function call. The OPERATE mode is set automatically by the DP master after the first DP application has logged on.

Example

The DP slaves connected to the DP bus form various functional groups. To control each group, a separate DP application is created under Windows. Each DP application can be started or stopped at any time.

☞

Via the DP programming interface, the DP application can activate or deactivate the AUTOCLEAR function (see also Section 3.7.6 and 5.4). This function changes the DP master from the OPERATE mode to the CLEAR mode automatically (see Section 4.8.3) if an error occurs on one of the configured slaves, in other words communication with the slave is no longer possible. To change back to the OPERATE mode, the user must send a job to change the mode. This is possible using the dpn_set_mode() function. Since such a call is reserved for central DP applications, the AUTOCLEAR function is only effective in a system with a central DP application.

☞

If no central DP application exists, the AUTOCLEAR function is not effective. In this case, the master does not change to the CLEAR mode, even if this function is configured using COML DP.

5.1.4 Constants for the "reference.access" Structure Element

The following constants are available in dpn_user.h for setting the DP application environment and the DP application type in the reference.access structure element:

Constant	DP Application Environment	DP Application Type
DPN_SYS_ CENTRAL	One of the DP applications is a central application.	
DPN_SYS_ NOT_CENTRAL	None of the DP applications is a central application.	
DPN_ROLE_ CENTRAL		The DP application that is logging on is a central application.
DPN_ROLE_ NOT_CENTRAL		The DP application that is logging on is not a central application.

Permitted Combinations

The values for the DP application environment and DP application type must be combined using an OR logic operation by the DP application before the entry is made in the reference.access structure element. The following combinations are permitted:

Combination	Meaning
(DPN_SYS_CENTRAL) | (DPN_ROLE_NOT_CENTRAL)	The DP application that is logging on is not a central DP application. A different DP application that also accesses the same CP takes over the control of the DP communication as the central application.
(DPN_SYS_CENTRAL) | (DPN_ROLE_CENTRAL)	The DP application that is logging on takes over the control of the CP communication as the central application.
(DPN_SYS_NOT_CENTRAL) | (DPN_ROLE_NOT_CENTRAL)	All the DP applications that access the CP have the same rank. There is no higher ranking application that controls the DP communication centrally.

Two Rules for Multitasking Applications

If several DP applications access the **same** CPU, two rules must be adhered to:

Rule 1: Only one of the applications can log on as the central DP application.

Rule 2: All the DP applications must enter the same value as the DP application environment (either DPN_SYS_CENTRAL or DPN_SYS_NOT_CENTRAL).

5.2 Monitoring Activity of the DP Application

Call Function

unsigned short int dpn_wd (struct dpn_interface far * ptr)

Description

This function starts an activity check (watchdog function) of the DP application on the CP. When the function is active and the DP application no longer accesses the DP programming interface as a result of a fault or error, this is detected by the CP. The CP then only sends data with the value 0h to the DP slaves **assigned** to this DP application.

Which Slaves are Assigned to the Application?

Only the slaves for which the application has the DPN_SLV_WRITE_READ access rights are assigned to the application.

☞

Without this precaution, a dangerous status could arise in the plant.

Reason:

The CP would continue to send the last output data to the DP cyclically although the DP application is no longer functioning correctly. If this response is not required, the DP application must call the dpn_wd() function before the productive phase starts.

Defaults

As default, the watchdog function is inactive. Using the dpn_wd() function, the watchdog can be activated or deactivated after the DP initialization (dpn_init() function call).

Effect of the Call

The watchdog time is entered as a call parameter in the user_data[0] structure element. If the call is successful, the CP checks whether the DP application executes DP function calls within the preset watchdog time. If this is the case, the watchdog is reset with each DP function call. If the DP application does not execute any DP function call during the preset watchdog time, the timer expires.

Timeout

If the DP application attempts to send output data or a control frame to DP slaves after the watchdog timer has expired, the job is acknowledged negatively until the DP application sends a new dpn_wd() call.

5.2.1 Call Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> stat_nr	irrelevant
ptr -> **length**	**>= 1**
ptr -> error_code	irrelevant
ptr -> slv_state	irrelevant
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> **user_data[0]**	**Watchdog time**

reference

The reference structure element must contain the handle returned with the dpn_init() function call.

length

Here a value >= 1 must be entered.

user_data[ ]

The watchdog time is entered in the user_data[0] structure element. If the watchdog time 0 is entered, the watchdog monitoring is deactivated.

Granularity of the Watchdog Time

One time unit corresponds to 400 ms. The accuracy of the measurement is 1 time unit.

Example: The value 10 is entered in the user_data[0] structure element. The watchdog expires after approximately 4 seconds if the DP application does not execute a DP function call within this time.

5.2.2 Return Parameters

ptr -> reference	unchanged
ptr -> stat_nr	irrelevant
ptr -> length	unchanged
ptr -> **error_code**	**Error identifier**
ptr -> slv_state	irrelevant
ptr -> **sys_state**	**Mode of the DP master**
ptr -> **sys_event**	**Event messages**
ptr -> user_data[ ]	unchanged

error_code

The error_code structure element is identical to the return parameter of the function call. The remaining return parameters of the dpn_interface structure are only valid if no error occurred (DPN_NO_ERROR).

The possible error identifiers are described in Section 4.4.

sys_state

The sys_state structure element contains the current mode of the DP master.

The possible modes are described in Section 4.8.3.

sys_event

The sys_event structure element identifies event messages during DP communication. The possible event messages are described in Sections 3.4 and 4.8.4.

Event Message "Watchdog Time Expired"

The WATCHDOG (monitoring time expired) identifier is entered in the sys_event structure element of the DP function calls if the DP application does not execute any DP function call within the preset watchdog time.

Note: The “watchdog time expired” event message is generated after a timeout has been detected. The message is then entered with all further DP calls. The entry is made until the DP application resets the event message.

Resetting the Event Message

To reset the “watchdog time expired” event message, the DP application must once again execute a dpn_wd() function.

5.3 Read Bus Parameters

Call function	**unsigned short int**	**dpn_read_bus_par (struct dpn_interface far * ptr)**

Description

With this function, a DP application can read out the current bus parameters of the CP. The data format of the bus parameters is described in Chapter 6.

5.3.1 Call Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> stat_nr	irrelevant
ptr -> **length**	**255**
ptr -> error_code	irrelevant
ptr -> slv_state	irrelevant
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> user_data[ ]	irrelevant

reference

The handle returned with the dpn_init() function call must be entered in the reference structure element.

length

Length of the user_data[ ] array. The value 255 must be entered.

5.3.2 Return Parameters

ptr -> reference	unchanged
ptr -> stat_nr	irrelevant
ptr -> **length**	**Number of valid bytes in user_data[ ]**
ptr **-> error_code**	**Error identifier**
ptr -> slv_state	irrelevant
ptr **-> sys_state**	**Mode of the DP master**
ptr **-> sys_event**	**Event messages**
ptr -> **user_data[ ]**	**Bus parameters**

length

The length structure element contains the size (number of bytes) of the bus parameters of the DP master entered in user_data[ ].

error_code

The error_code structure element is identical to the return parameter of the function call. The remaining return parameters of the dpn_interface structure are only valid if no error occurred (DPN_NO_ERROR).

The possible error identifiers are described in Section 4.4.

sys_state

The sys_state structure element contains the current mode of the DP master.
The possible modes are described in Section 4.8.3.

sys_event

The structure element sys_event identifies event messages during DP communication. The possible event messages are described in Sections 3.4 and 4.8.4.

user_data[ ]

This structure element contains the bus parameters of the DP master. Chapter 6 describes the structure of the bus parameters. The structure definition of the bus parameters can be found in the dpn_user.h include file (dpn_buspar structure).

5.4 Write Bus Parameters

Call function **unsigned short int** **dpn_load_bus_par (struct dpn_interface far * ptr)**

Description

With this function, the bus parameter data record of the DP master can be modified by a central application. The structure of the bus parameters transferred is described in Chapter 6. The DP firmware only evaluates the following DP-specific parts of the bus parameters:

- ➢ POLL_TIMEOUT
- ➢ BP_FLAG

Before the function dpn_load_bus_par() is used, the current bus parameters must be read into the dpn_bus_par structure with the function dpn_read_bus_par(). The parameters POLL_TIMEOUT and BP_FLAG can then be modified in the dpn_bus_par structure.

POLL_TIMEOUT

If there is a service request from a DP master Class 2 to the CP, this value specifies the time within which the response must be fetched by the DP master Class 2.

Time unit: 50 ms

BP_FLAG

The BP_FLAG is used to activate or deactivate the autoclear function of the DP master.

Activate: OR logic operation of the BpFlag structure element with the value 80h.

Deactivate: AND logic operation of the BpFlag structure element with the value 7Fh.

☞ **Bus parameters can only be written when the DP master is in the OFFLINE mode and they can only be written by a central DP application.**

5.4.1 Call Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> stat_nr	irrelevant
ptr -> **length**	**>= 36**
ptr -> error_code	irrelevant
ptr -> slv_state	irrelevant
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> **user_data[ ]**	**Bus parameters**

reference The handle returned with the dpn_init() function call must be entered in the reference structure element.

length The length structure element contains the number of valid bytes of the user_data[ ] array. A minimum value of 36 must be entered.

user_data[ ] The element user_data[ ] structure element contains the bus parameters. The structure of the bus parameters is described in Chapter 6. The structure definition of the bus parameters is in the dpn_user.h include file (dpn_buspar structure).

5.4.2 Return Parameters

ptr -> reference	unchanged
ptr -> stat_nr	irrelevant
ptr -> length	unchanged
ptr -> **error_code**	**Error identifier**
ptr -> slv_state	irrelevant
ptr -> **sys_state**	**Mode of the DP master**
ptr -> **sys_event**	**Event messages**
ptr -> user_data[ ]	unchanged

error_code

The error_code structure element is identical to the return parameter of the function call. The remaining return parameters of the dpn_interface structure are only valid if no error occurred (DPN_NO_ERROR).

The possible error identifiers are described in Section 4.4.

sys_state

The sys_state structure element contains the current mode of the DP master. The possible modes are described in Section 4.8.3.

sys_event

The structure element sys_event identifies event messages during DP communication. The possible event messages are described in Sections 3.4 and 4.8.4.

5.5 Read Slave Parameters

Call Function **unsigned short int dpn_read_slv_par (struct dpn_interface far * ptr)**

Description

With this function, a DP application can read out the various components of the slave parameters of a DP slave. The data format of the individual parts of the slave parameters corresponds to the format described in Chapter 6. Evaluation of the slave parameters requires detailed knowledge of the PROFIBUS DP standard. For this reason, this function is only intended for specialized diagnostic purposes.

The following components of the slave parameters can be selected using the slv_state structure element:

- ➢ Sl-Flag, Slave-Type, Octet-String
- ➢ Parameter assignment data
- ➢ Configuration data
- ➢ Add_Tab list
- ➢ Slave_User_Data

☞ **This service can only be used with standard slaves.**

Structure of the Slave Parameters

The structure of the slave parameters is described in Chapter 6.

5.5.1 Call Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> **stat_nr**	**Address of the DP slave**
ptr -> **length**	**255**
ptr -> error_code	irrelevant
ptr -> **slv_state**	**Component selection**
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> user_data[0]	irrelevant

reference The handle returned with the dpn_init() function call must be entered in the reference structure element.

stat_nr The stat_nr structure element contains the L2 address of the selected DP slave.

length The value 255 must be entered.

slv_state The slv_state structure element contains the information defining which part of the slave parameters must be read out. The job-defining values are contained in the dpn_user.h include file.

Job-defining Value	Used to Select..
DPN_SLV_PARA_TYP	Sl-Flag, Slave-type and Octet-String
DPN_SLV_PARA_PRM_DATA	Parameter assignment data
DPN_SLV_PARA_CFG_DATA	Configuration data
DPN_SLV_PARA_ADD_TAB	Add_Tab - list
DPN_SLV_PARA_USER_DATA	Slave_User_Data

5.5.2 Return Parameters

ptr -> reference	unchanged
ptr -> stat_nr	unchanged
ptr -> **length**	**Number of valid bytes in user_data[ ]**
ptr **-> error_code**	**Error identifier**
ptr -> **slv_state**	**Operating status of the DP slave**
ptr **-> sys_state**	**Mode of the DP master**
ptr **-> sys_event**	**Event messages**
ptr -> **user_data[ ]**	**Slave parameters**

length

The length structure element contains the size (number of bytes) of the components of the slave parameters entered in user_data[ ].

error_code

The error_code structure element is identical to the return parameter of the function call. The remaining return parameters of the dpn_interface structure are only valid if no error occurred (DPN_NO_ERROR).

The possible error identifiers are described in Section 4.4.

slv_state

The structure element slv_state contains the current operating status of the DP slave.

The possible operating statuses are described in Section 4.8.2.

sys_state

The sys_state structure element contains the current mode of the DP master.

The possible modes are described in Section 4.8.3.

sys_event

The structure element sys_event identifies event messages during DP communication.
The possible event messages are described in Sections 3.4 and 4.8.4.

user_data[ ]

This structure element contains the selected components of the slave parameters.

5.6 Activating/Deactivating a DP Slave

Call Function **unsigned short int** **dpn_set_slv_state (struct dpn_interface far * ptr)**

Description

With this function, the operating status of a DP slave can be modified while the DP application is running.

The function makes it possible to activate or deactivate the DP slave dynamically.

The mode of a DP slave can only be modified by a DP application when the identifier DPN_SLV_WRITE_READ has been entered as the access right for the dpn_init() call.

5.6.1 Call Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> **stat_nr**	**Address of the DP slave**
ptr -> **length**	**>= 1**
ptr -> error_code	irrelevant
ptr -> slv_state	irrelevant
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> **user_data[0]**	**New mode of the DP slave**

reference

The handle returned with the dpn_init() function call must be entered in the reference structure element.

stat_nr

The stat_nr structure element contains the L2 address of the selected DP slave.

length

The length structure element contains the number of valid bytes of the user_data[ ] array.

A minimum value of 1 must be entered.

user_data[ ]

The user_data[0] structure element contains the new mode of the DP slave. The value for the mode is contained in the dpn_user.h include file.

Mode	Effect
DPN_SLV_ACTIVATE	The slave is activated in the local database. Following this, the DP master attempts to assign parameters to the slave, to configure it and to transfer data. (Condition: The master is in the CLEAR or OPERATE mode.)
DPN_SLV_DEACTIVATE	The slave is deactivated in the local database, in other words there is no data exchange with the slave.

5.6.2 Return Parameters

ptr -> reference	unchanged
ptr -> stat_nr	unchanged
ptr -> length	unchanged
ptr -> **error_code**	**Error identifier**
ptr -> **slv_state**	**Operating status of the DP slave**
ptr -> **sys_state**	**Mode of the DP master**
ptr -> **sys_event**	**Event messages**
ptr -> user_data[ ]	unchanged

error_code

The error_code structure element is identical to the return parameter of the function call. The remaining return parameters of the dpn_interface structure are only valid if no error occurred (DPN_NO_ERROR).

The possible error identifiers are described in Section 4.4.

slv_state

The slv_state structure element contains the current operating status of the DP slave.

The possible operating statuses are described in Section 4.8.2.

Note: To activate or deactivate a slave, various activities are required so that the indicated operating status of the DP slave can still be distinguished from the new setting.

sys_state

The sys_state structure element contains the current mode of the DP master.

The possible modes are described in Section 4.8.3.

sys_event

The structure element sys_event identifies event messages during DP communication.

The possible event messages are described in Sections 3.4 and 4.8.4.

5.7 Querying the Configuration of the DP System

Call Function	**unsigned short int**	**dpn_read_cfg (struct dpn_interface far * ptr)**

Description

With this function, a DP application can find out the total configuration of the DP database. The function provides information about:

- ➢ the number of configured DP slaves
- ➢ the type of configured DP slaves.

5.7.1 Call Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> stat_nr	irrelevant
ptr -> **length**	**>= 126**
ptr -> error_code	irrelevant
ptr -> slv_state	irrelevant
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> user_data[ ]	irrelevant

reference

The handle returned with the dpn_init() function call must be entered in the reference structure element.

length

The structure element length contains the number of valid bytes of the user_data[ ] array.

A minimum value of 126 must be entered.

5.7.2 Return Parameters

ptr -> reference	unchanged
ptr -> stat_nr	unchanged
ptr -> **length**	**Number of valid bytes in user_data[ ]**
ptr -> **error_code**	**Error identifier**
ptr -> slv_state	irrelevant
ptr -> **sys_state**	**Mode of the DP master**
ptr -> **sys_event**	**Event messages**
ptr -> **user_data[ ]**	**Configuration of the DP system**

length

The length structure element contains the size of the entire configuration in user_data[ ].

error_code

The error_code structure element is identical to the return parameter of the function call. The remaining return parameters of the dpn_interface structure are only valid if no error occurred (DPN_NO_ERROR).

The possible error identifiers are described in Section 4.4.

sys_state

The sys_state structure element contains the current mode of the DP master.

The possible modes are described in Section 4.8.3.

sys_event

The structure element sys_event identifies event messages during DP communication.

The possible event messages are described in Sections 3.4 and 4.8.4.

user_data[ ]

Type information is entered in the user_data[ ] structure element for every possible slave address. Each entry is in a separate byte. The index within the array corresponds to the L2 address of the DP slave.

Example: user_data[2] contains the type information of the slave with the L2 address 2.

The following type information can be entered:

Possible Entries	Meaning
DPN_CFG_NO_SLV	No DP slave
DPN_CFG_NORM	Standard DP slave
DPN_CFG_ET200_U	Non-standard slave: ET 200U
DPN_CFG_ET200K_B	Non-standard slave: ET 200K/B
DPN_CFG_ET200_SPM	Non-standard slave: General SPM station

5.8 Requesting the Diagnostic Data of a Slave

Call Function	**unsigned short int**	**dpn_slv_diag (struct dpn_interface far * ptr)**

Description

With this function, a DP application can request the diagnostic data of a DP slave. The diagnostic data are read out of the local diagnostic buffer. Whether or not diagnostic data exist, can be determined from the slv_state structure element of a previous DP function call. After the diagnostic data have been read out, the corresponding ID is reset in slv_state (provided no new diagnostic data have been added in the meantime).

Management of the diagnostic ID is application-related. This means that if DP application "A" reads out the diagnostic data, the diagnostic ID for DP application "B" is **not** reset.

5.8.1 Call Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> **stat_nr**	**Address of the DP slave**
ptr -> **length**	**255**
ptr -> error_code	irrelevant
ptr -> slv_state	irrelevant
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> user_data[ ]	irrelevant

reference

The handle returned with the dpn_init() function call must be entered in the reference structure element.

stat_nr

The stat_nr structure element contains the L2 address of the selected DP slave.

length

The structure element length contains the number of valid bytes of the user_data[ ] array.

The value 255 must be entered.

5.8.2 Return Parameters

ptr -> reference	unchanged
ptr -> stat_nr	unchanged
ptr -> **length**	**Number of valid bytes in user_data[ ]**
ptr -> **error_code**	**Error identifier**
ptr -> **slv_state**	**Operating status of the DP slave**
ptr -> **sys_state**	**Mode of the DP master**
ptr -> **sys_event**	**Event messages**
ptr -> **user_data[ ]**	**Diagnostic data**

length

The length structure element contains the size (number of bytes) of the diagnostic data in user_data[ ].

error_code

The error_code structure element is identical to the return parameter of the function call. The remaining return parameters of the dpn_interface structure are only valid if no error occurred (DPN_NO_ERROR).

The possible error identifiers are described in Section 4.4.

slv_state

The structure element slv_state contains the current operating status of the DP slave.

The possible operating statuses are described in Section 4.8.2.

sys_state

The sys_state structure element contains the current mode of the DP master.

The possible modes are described in Section 4.8.3.

sys_event

The structure element sys_event identifies event messages during DP communication.

The possible event messages are described in Sections 3.4 and 4.8.4.

user_data[ ]

The structure element user_data[ ] contains the diagnostic data of the DP slave.

The structure of the diagnostic data is described in Chapter 6.

5.9 Reading Status Information of the DP System

Call Function	**unsigned short int**	**dpn_read_sys_info (struct dpn_interface far * ptr)**

Description

This function provides an overview of the current status of the DP slaves known to the CP. The function also indicates whether diagnostic data exist for each DP slave.

5.9.1 Call Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> stat_nr	irrelevant
ptr -> **length**	**>=126**
ptr -> error_code	irrelevant
ptr -> slv_state	irrelevant
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> user_data[ ]	irrelevant

reference

The handle returned with the dpn_init() function call must be entered in the reference structure element.

length

The length structure element contains the number of valid bytes of the user_data[ ] array.

A minimum value of 126 must be entered.

5.9.2 Return Parameters

ptr -> reference	unchanged
ptr -> stat_nr	unchanged
ptr -> **length**	**Number of valid bytes in user_data[ ]**
ptr -> **error_code**	**Error identifier**
ptr -> slv_state	irrelevant
ptr -> **sys_state**	**Mode of the DP master**
ptr -> **sys_event**	**Event messages**
ptr -> **user_data[ ]**	**Status information**

length

The length structure element contains the size of the status information in user_data[ ].

error_code

The error_code structure element is identical to the return parameter of the function call. The remaining return parameters of the dpn_interface structure are only valid if no error occurred (DPN_NO_ERROR).

The possible error identifiers are described in Section 4.4.

sys_state

The sys_state structure element contains the current mode of the DP master.

The possible modes are described in Section 4.8.3.

sys_event

The structure element sys_event identifies event messages during DP communication.

The possible event messages are described in Sections 3.4 and 4.8.4.

user_data[ ]

The status information for every possible slave address is entered in user_data[ ]. Each entry is made in a separate byte. The index in the array corresponds to the L2 address of the DP slave.

Example: the status of the slave with address 2 is entered in user_data[2].

The following status information is possible:

Status Information	Meaning
DPN_SLV_STAT_READY	The DP slave is in the data transfer phase.
DPN_SLV_STAT_READY_DIAG	The DP slave is in the data transfer phase, and diagnostic data exist.
DPN_SLV_STAT_OFFLINE	The DP slave is not in the data transfer phase (CP startup).
DPN_SLV_STAT_NOT_READY	The DP slave is not in the data transfer phase.
DPN_SLV_STAT_NOT_READY_DIAG	The DP slave is not in the data transfer phase, and diagnostic data exist.
DPN_SLV_STAT_NOT_ACTIVE	The DP slave is not activated.

5.10 Sending Output Data to a DP Slave

Call Function	**unsigned short int dpn_out_slv (struct dpn_interface far * ptr)**
Description	With this function, a DP application can transfer output data to a DP slave. The L2 address and the length of the output data are compared with the configured values in the DP database. The data are only accepted if both parameters match. The output data are entered in the internal data memory of the CP.

5.10.1 Call Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> **stat_nr**	**Address of the DP slave**
ptr -> **length**	**Length of the output data**
ptr -> error_code	irrelevant
ptr -> slv_state	irrelevant
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> **user_data[ ]**	**Output data**

reference	The handle returned with the dpn_init() function call must be entered in the reference structure element.
stat_nr	The stat_nr structure element contains the L2 address of the selected DP slave.
length	The length structure element contains the number of output data in the user_data[ ] array.
user_data[ ]	The output data are entered in the user_data[ ] structure element. The format of the output data is described in Chapter 6.

5.10.2 Return Parameters

ptr -> reference	unchanged
ptr -> stat_nr	unchanged
ptr -> length	unchanged
ptr -> **error_code**	**Error identifier**
ptr -> **slv_state**	**Operating status of the DP slave**
ptr -> **sys_state**	**Mode of the DP master**
ptr -> **sys_event**	**Event messages**
ptr -> user_data[ ]	unchanged

error_code

The error_code structure element is identical to the return parameter of the function call. The remaining return parameters of the dpn_interface structure are only valid if no error occurred (DPN_NO_ERROR).

The possible error identifiers are described in Section 4.4.

☞

The entry of DPN_NO_ERROR in the error_code structure element simply means that the output data could be entered in the local data buffer of the CP.

Whether or not the output data are sent to the DP slave must be determined by evaluating the structure elements slv_state (operating status of the DP slave) and sys_state (mode of the DP master).

The DP slave only receives the output data of the DP master when the DP slave is in the productive phase. This is the case when the slv_state structure element contains one of the following values:

- **DPN_SLV_STAT_READY or**
- **DPN_SLV_STAT_READY_DIAG**

The DP master must also be in the OPERATE mode, in other words the sys_state structure element must have the value

- **DPN_SYS_OPERATE**

slv_state

The structure element slv_state contains the current operating status of the DP slave.

The possible operating statuses are described in Section 4.8.2.

sys_state

The sys_state structure element contains the current mode of the DP master.

The possible modes are described in Section 4.8.3.

sys_event

The structure element sys_event identifies event messages during DP communication.

The possible event messages are described in Sections 3.4 and 4.8.4.

5.11 Sending Output Data to Several DP Slaves

Call Function **unsigned short int** **dpn_out_slv_m**
(struct dpn_interface_m far * ptr)

Description

With this function, a DP application can send output data to **several** DP slaves with a single function call. Within the **dpn_interface_m** structure, a **dpn_interface_s** structure exists for every DP slave. The structure of dpn_interface_s corresponds to that of the dpn_interface structure.

When using the function, remember the following differences compared with the dpn_out_slv() function:

- ➢ The output data sent to one slave must not exceed 32 bytes.
- ➢ A maximum of 32 slaves can be addressed.
- ➢ The slaves addressed must be obtainable using the same CP.
- ➢ Unused structure elements (dpn_interface_s) must be marked as invalid by the entry DPN_IF_S_UNUSED in the stat_nr structure element.

The return parameters of the function call indicate whether an error occurred in one or more structure elements. If a error occurred (return value ≠ DPN_NO_ERROR), the return parameter contains the error ID of the first structure element with an error.

5.11.1 Call Parameters

ptr -> **dpn_if_single[x].reference**	**Handle of the DP application (only relevant in the first valid structure)**
ptr -> **dpn_if_single[x].stat_nr**	**Address of the DP slave or DPN_IF_S_UNUSED;**
ptr -> **dpn_if_single[x].length**	**Number of valid bytes in the user_data[ ] array (Maximum: 32 bytes)**
ptr -> dpn_if_single[x].error_code	irrelevant
ptr -> dpn_if_single[x].slv_state	irrelevant
ptr -> dpn_if_single[x].sys_state	irrelevant
ptr -> dpn_if_single[x].sys_event	irrelevant
ptr -> **dpn_if_single[x].user_data[ ]**	**Output data**

reference
The handle returned with the dpn_init() function call must be entered in the reference structure element. The handle only needs to be entered in the first dpn_interface_s structure.

stat_nr
The stat_nr structure element contains the L2 address of the selected DP slave.

☞ **If a dpn_interface_s structure is not used, the constant DPN_IF_S_UNUSED must be entered here. This entry marks the structure as invalid.**

length
The length structure element contains the number of output data in the user_data[ ] array.

user_data[ ]
The output data are entered in the user_data[ ] structure.

The format of the output data is described in Chapter 6.

5.11.2 Return Parameters

ptr -> dpn_if_single[x].reference	irrelevant
ptr -> dpn_if_single[x].stat_nr	unchanged
ptr -> dpn_if_single[x].length	unchanged
ptr -> **dpn_if_single[x].error_code**	**Error identifier**
ptr -> **dpn_if_single[x].slv_state**	**Status of the DP slave**
ptr -> **dpn_if_single[x].sys_state**	**Mode of the DP master**
ptr -> **dpn_if_single[x].sys_event**	**Event messages**
ptr -> dpn_if_single[x].user_data[ ]	unchanged

error_code

The error_code structure element indicates the result of the job processing. This is only valid when the return parameter of the function call is DPN_NO_ERROR. The remaining return parameters of the dpn_interface_s structure are only valid if no error occurred (DPN_NO-ERROR).

The possible error identifiers are described in Section 4.4.

☞

The entry of DPN_NO_ERROR in the error_code structure element simply means that the output data could be entered in the local data buffer of the CP.

Whether or not the output data are sent to the DP slave must be determined by evaluating the structure elements slv_state (operating status of the DP slave) and sys_state (mode of the DP master).

The DP slave only receives the output data of the DP master when the DP slave is in the productive phase. This is the case when the slv_state structure element contains one of the following values:

- ➢ **DPN_SLV_STAT_READY or**
- ➢ **DPN_SLV_STAT_READY_DIAG**

The DP master must also be in the OPERATE mode, in other words the sys_state structure element must have the value

- ➢ **DPN_SYS_OPERATE**

slv_state

The structure element slv_state contains the current operating status of the DP slave.

The possible operating statuses are described in Section 4.8.2.

sys_state

The sys_state structure element contains the current mode of the DP master.

The possible modes are described in Section 4.8.3.

sys_event The structure element sys_event identifies event messages during DP communication.

The possible event messages are described in Sections 3.4 and 4.8.4.

5.11.3 Return Value of the Function Call

The return value of the function indicates whether an error occurred with one or more of the dpn_interface_s structures. The function could only be processed by the CP if no error occurred (DPN_NO_ERROR). **If an error occurred, processing is aborted.** In this case, output data may possibly already have been passed on to individual DP slaves.

If no error occurred, the return parameters of the individual dpn_interface_s structures must be evaluated.

5.12 Reading the Local Output Data of a DP Slave

Call Function	**unsigned short int**	**dpn_read_slv (struct dpn_interface far * ptr)**

Description

With this function, a DP application can read out the current output data entered in the local data buffer of the CP.

The function primarily supports communication between the tasks of a multitasking operating system when the tasks access the same CP. The function allows a task "A" to determine which output data another task "B" sends to a DP slave.

5.12.1 Call Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> **stat_nr**	**Address of the DP slave**
ptr -> **length**	**255**
ptr -> error_code	irrelevant
ptr -> slv_state	irrelevant
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> user_data[ ]	irrelevant

reference

The handle returned with the dpn_init() function call must be entered in the reference structure element.

stat_nr

The stat_nr structure element contains the L2 address of the selected DP slave.

length

The structure element length contains the size of the user_data[ ] structure element.

Here, the value 255 must be entered.

5.12.2 Return Parameters

ptr -> reference	unchanged
ptr -> stat_nr	unchanged
ptr -> **length**	**Number of output data**
ptr **-> error_code**	**Error identifier**
ptr -> **slv_state**	**Operating status of the DP slave**
ptr **-> sys_state**	**Mode of the DP master**
ptr **-> sys_event**	**Event messages**
ptr -> **user_data[ ]**	**Output data to the DP slave**

length

The length structure element contains the number of output data in the user_data[ ] structure element.

error_code

The error_code structure element is identical to the return parameter of the function call. The remaining return parameters of the dpn_interface structure are only valid if no error occurred (DPN_NO_ERROR).

The possible error identifiers are described in Section 4.4.

☞

The entry of DPN_NO_ERROR in the error_code structure element simply means that the output data could be entered in the local data buffer of the CP.

Whether or not the output data are sent to the DP slave must be determined by evaluating the structure elements slv_state (operating status of the DP slave) and sys_state (mode of the DP master).

The DP slave only receives the output data of the DP master when the DP slave is in the productive phase. This is the case when the slv_state structure element contains one of the following values:

- **DPN_SLV_STAT_READY or**
- **DPN_SLV_STAT_READY_DIAG**

The DP master must also be in the OPERATE mode, in other words the sys_state structure element must have the value

- **DPN_SYS_OPERATE**

slv_state

The structure element slv_state contains the current operating status of the DP slave.

The possible operating statuses are described in Section 4.8.2.

sys_state

The sys_state structure element contains the current mode of the DP master.

The possible modes are described in Section 4.8.3.

sys_event The structure element sys_event identifies event messages during DP communication.

The possible event messages are described in Sections 3.4 and 4.8.4.

user_data[ ] The user_data[ ] structure element contains the output data for the DP slave.

5.13 Reading the Input Data of a DP Slave

Call Function **unsigned short int dpn_in_slv (struct dpn_interface far * ptr)**

Description With this function, a DP application can read out the current input data of a DP slave. The data are taken from the local data buffer of the CP. The image of the data buffer is updated cyclically as long as the DP master is in one of the modes CLEAR or OPERATE.

5.13.1 Call Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> **stat_nr**	**Address of the DP slave**
ptr -> **length**	**255**
ptr -> error_code	irrelevant
ptr -> slv_state	irrelevant
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> user_data[ ]	irrelevant

reference The handle returned with the dpn_init() function call must be entered in the reference structure element.

stat_nr The stat_nr structure element contains the L2 address of the selected DP slave.

length The length structure element contains the size of the structure element user_data[ ].

Here, the value 255 must be entered.

5.13.2 Return Parameters

ptr -> reference	unchanged
ptr -> stat_nr	unchanged
ptr -> **length**	**Number of input bytes**
ptr **-> error_code**	**Error identifier**
ptr -> **slv_state**	**Operating status of the DP slave**
ptr **-> sys_state**	**Mode of the DP master**
ptr **-> sys_event**	**Event messages**
ptr -> **user_data[ ]**	**Input data of the DP slave**

length

The length structure element contains the number of input data in the user_data[ ] structure element.

error_code

The error_code structure element is identical to the return parameter of the function call. The remaining return parameters of the dpn_interface structure are only valid if no error occurred (DPN_NO_ERROR).

The possible error identifiers are described in Section 4.4.

☞

The entry of DPN_NO_ERROR in the error_code structure element simply means that the input data could be entered in the local data buffer of the CP.

Whether the input data of the DP slave are valid must be determined by evaluating the structure elements slv_state (operating status of the DP slave) and sys_state (mode of the DP master).

The slave only sends the input data to the DP master when it is in the productive phase. If this is the case when the slv_state structure element contains one of the following values:

➢ **DPN_SLV_STAT_READY or**

➢ **DPN_SLV_STAT_READY_DIAG**

The DP master must also be in one of the modes CLEAR or OPERATE, in other words the sys_state structure element must have one of the following values:

➢ **DPN_SYS_CLEAR or**

➢ **DPN_SYS_OPERATE**

slv_state

The structure element slv_state contains the current operating status of the DP slave.

The possible operating statuses are described in Section 4.8.2.

sys_state The sys_state structure element contains the current mode of the DP master.

The possible modes are described in Section 4.8.3.

sys_event The structure element sys_event identifies event messages during DP communication.

The possible event messages are described in Sections 3.4 and 4.8.4.

user_data[ ] The structure element user_data[ ] contains the input data of the DP slave.

The format of the input data is described in Chapter 6.

5.14 Reading the Input Data of Several DP Slaves

Call Function

unsigned short int dpn_in_slv_m (struct dpn_interface_m far * ptr)

Description

With this function, a DP application can read out the current input data of **several** DP slaves. The application receives an image of the local data buffer of the CP. The image is updated cyclically.

Within the dpn_interface_m structure, there is a dpn_interface_s structure for every DP slave. The structure of the dpn_interface_s structure corresponds to that of the dpn_interface structure.

When using the function, remember the following differences compared with the dpn_in_slv() function call:

➢ The input data of a slave must not exceed 32 bytes.

➢ The addressed slaves must be obtainable via the same CP.

➢ A maximum of 32 slaves can be addressed.

➢ Unused structure elements (dpn_interface_s) must be marked as invalid using the DPN_IF_S_UNUSED entry in the stat_nr structure element.

From the return value of the function call, it is possible to recognize whether an error occurred with a single structure element. If an error occurs (return value ≠ DPN_NO_ERROR), the return value contains the error ID of the first incorrect structure element. The processing of the function call is aborted if an error is detected in a structure element.

5.14.1 Values of the "dpn_interface_s" Structures

ptr -> **dpn_if_single[x].reference**	**Handle of the DP application (only relevant in the first valid structure)**
ptr -> **dpn_if_single[x].stat_nr**	**Address of the DP slave or DPN_IF_S_UNUSED;**
ptr -> dpn_if_single[x].length	irrelevant
ptr -> dpn_if_single[x].error_code	irrelevant
ptr -> dpn_if_single[x].slv_state	irrelevant
ptr -> dpn_if_single[x].sys_state	irrelevant
ptr -> dpn_if_single[x].sys_event	irrelevant
ptr -> dpn_if_single[x].user_data[ ]	irrelevant

reference The handle returned with the dpn_init() function call must be entered in the reference structure element. The handle only needs to be entered in the first valid dpn_interface_s structure.

stat_nr The stat_nr structure element contains the L2 address of the selected DP slave.

☞ **If a dpn_interface_s structure is not used, the constant “DPN_IF_S_UNUSED“ must be entered here. This entry marks the structure as invalid.**

5.14.2 Return Parameters of the "dpn_interface_s" Structures

ptr -> dpn_if_single[x].reference	unchanged
ptr -> dpn_if_single[x].stat_nr	unchanged
ptr -> **dpn_if_single[x].length**	**Number of valid bytes in user_data[ ]**
ptr -> **dpn_if_single[x].error_code**	**Error identifier**
ptr -> **dpn_if_single[x].slv_state**	**Status of the DP slave**
ptr -> **dpn_if_single[x].sys_state**	**Mode of the DP master**
ptr -> **dpn_if_single[x].sys_event**	**Event messages**
ptr -> **dpn_if_single[x].user_data[ ]**	**Input data of the DP slave**

length

The length structure element contains the number of input data in the user_data[ ] structure element.

error_code

The error_code structure element indicates the result of the job processing. This is only valid when the return parameter of the function call is DPN_NO_ERROR. The remaining return parameters of the dpn_interface_s structure are only valid when no error occurred (DPN_NO_ERROR).

The possible error identifiers are described in Section 4.4.

☞

The entry of DPN_NO_ERROR in the error_code structure element simply means that the input data could be entered in the local data buffer of the CP.

Whether the input data of the DP slave are valid must be determined by evaluating the structure elements slv_state (operating status of the DP slave) and sys_state (mode of the DP master).

The slave only sends the input data to the DP master when it is in the productive phase. If this is the case when the slv_state structure element contains one of the following values:

- **DPN_SLV_STAT_READY or**
- **DPN_SLV_STAT_READY_DIAG**

The DP master must also be in one of the modes CLEAR or OPERATE, in other words the sys_state structure element must have one of the following values:

- **DPN_SYS_CLEAR or**
- **DPN_SYS_OPERATE**

slv_state

The structure element slv_state contains the current operating status of the DP slave.

The possible operating statuses are described in Section 4.8.2.

sys_state

The sys_state structure element contains the current mode of the DP master.

The possible modes are described in Section 4.8.3.

sys_event

The structure element sys_event identifies event messages during DP communication.

The possible event messages are described in Sections 3.4 and 4.8.4.

user_data[ ]

The structure element user_data[ ] contains the input data of the DP slave.

The format of the input data is described in Chapter 6.

5.14.3 Return Value of the Function

The return value indicates whether an error occurred in one or more dpn_interface_s structure elements. The CP was only able to process the function if no error occurred (DPN_NO_ERROR). **If an error occurs, processing is aborted.**

If no error occurred, the return parameters of the individual dpn_interface_s structures must be evaluated.

5.15 Setting the Mode of the DP Master

Call Function **unsigned short int dpn_set_mode (struct dpn_interface far * ptr)**

Description The mode of the DP master can be changed using this function. This function can only be called by a central application (see Section 5.1.3).

The following DP modes can be set:

Mode	Description
OFFLINE	There is no DP communication.
STOP	Communication possible only with a DP master Class 2. There is no communication with the DP slaves.
CLEAR	The DP slaves are assigned parameters and configured and data is transferred, in the output direction 0 bytes are sent to the DP slaves.
OPERATE	Productive data exchange with the DP slaves.

☞ **When setting a new mode, no modes must be skipped.** Starting from the current mode, the modes must be run through in the prescribed order OFFLINE -> STOP -> CLEAR -> OPERATE (in ascending or descending order). The current operating status can be read out using the dpn_get_mode() call.

Following a dpn_set_mode() call, the subsequent dpn_get_mode() calls must check whether the required mode has been set. Another new mode can only be set after the first one has been achieved.

Remember that there is **no** change to the OPERATE mode, if the AUTOCLEAR event is indicated in the sys_event return parameter.

Example of Maintaining the Correct Order You want to set the OPERATE mode. The dpn_get_mode() call determines that the current mode is STOP. By calling dpn_set_mode(), the CLEAR mode must first be set. The OPERATE mode can only be set after the CLEAR mode has been adopted.

5.15.1 Call Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> stat_nr	irrelevant
ptr -> length	irrelevant
ptr -> error_code	irrelevant
ptr -> slv_state	irrelevant
ptr -> **sys_state**	**New mode**
ptr -> sys_event	irrelevant
ptr -> user_data[0]	irrelevant

reference

The handle returned with the dpn_init() function call must be entered in the reference structure element.

sys_state

The structure element contains the new mode of the DP master.

Values in sys_state	Mode
DPN_SYS_OFFLINE	OFFLINE
DPN_SYS_STOP	STOP
DPN_SYS_CLEAR	CLEAR
DPN_SYS_OPERATE	OPERATE

5.15.2 Return Parameters

ptr -> reference	unchanged
ptr -> stat_nr	irrelevant
ptr -> length	irrelevant
ptr **-> error_code**	**Error identifier**
ptr -> slv_state	irrelevant
ptr **-> sys_state**	**Mode of the DP master**
ptr **-> sys_event**	**Event messages**
ptr -> user_data[ ]	irrelevant

error_code

The error_code structure element is identical to the return parameter of the function call. The remaining return parameters of the dpn_interface structure are only valid if no error occurred (DPN_NO_ERROR).

The possible error identifiers are described in Section 4.4.

sys_state

The sys_state structure element contains the mode of the DP master that was valid when the job was sent.

The possible modes are described in Section 4.8.3.

Note: Since setting a new mode requires certain activities by the DP master, the value determined may differ from the current set value.

sys_event

The structure element sys_event identifies event messages during DP communication.

The possible event messages are described in Sections 3.4 and 4.8.4.

5.16 Querying the Mode of the DP Master

Call Function **unsigned short int dpn_get_mode (struct dpn_interface far * ptr)**

Description The current mode of the DP master can be determined using this function.

The DP master can be in one of the following modes:

Mode	Description
OFFLINE	There is no DP communication
STOP	Communication possible only with a DP master Class 2. There is also no communication with DP slaves.
CLEAR	The DP slaves have parameters assigned and are configured and data is transferred, in the output direction 0 bytes are sent to the DP slaves.
OPERATE	Productive data exchange with the DP slaves.

5.16.1 Call Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> stat_nr	irrelevant
ptr -> length	irrelevant
ptr -> error_code	irrelevant
ptr -> slv_state	irrelevant
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> user_data[ ]	irrelevant

reference The handle returned with the dpn_init() function call must be entered in the reference structure element.

5.16.2 Return Parameters

ptr -> reference	unchanged
ptr -> stat_nr	irrelevant
ptr -> length	irrelevant
ptr **-> error_code**	**Error identifier**
ptr -> slv_state	irrelevant
ptr **-> sys_state**	**Mode of the DP master**
ptr **-> sys_event**	**Event messages**
ptr -> user_data[ ]	irrelevant

error_code

The error_code structure element is identical to the return parameter of the function call. The remaining return parameters of the dpn_interface structure are only valid if no error occurred (DPN_NO_ERROR).

The possible error identifiers are described in Section 4.4.

sys_state

The sys_state structure element contains the current mode of the DP master.

Return Parameters in sys_state	Mode
DPN_SYS_OFFLINE	OFFLINE
DPN_SYS_STOP	STOP
DPN_SYS_CLEAR	CLEAR
DPN_SYS_OPERATE	OPERATE

The possible modes are described in Section 4.8.3.

sys_event

The structure element sys_event identifies event messages during DP communication.

The possible event messages are described in Sections 3.4 and 4.8.4.

5.17 Acyclic Transmission of a Control Frame

Call Function **unsigned short int dpn_global_ctrl (struct dpn_interface far * ptr)**

Description With this function, a DP application can send control commands to one slave, a group of slaves, several groups or all DP slaves. The control commands are sent to the DP slaves using a broadcast or a multicast frame.

A broadcast frame is intended for **all** DP slaves.

A multicast frame is intended for **several** DP slaves.

The following control commands can be transferred with the function call:

Control Command	Effect on the DP slaves
FREEZE	The states of the inputs are read in and frozen.
UNFREEZE	The freezing of the inputs is canceled.
SYNC	Output is frozen.
UNSYNC	The UNSYNC command cancels the SYNC command.

☞ **Control commands are only effective with standard slaves.**

5.17.1 Call Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> **stat_nr**	**slave address**
ptr -> **length**	**>= 1**
ptr -> error_code	irrelevant
ptr -> **slv_state**	**Control_Command**
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> **user_data[0]**	**Group_identifier**

reference

The handle returned with the dpn_init() function call must be entered in the reference structure element.

stat_nr

The stat_nr structure element contains the L2 address of a single DP slave or the global address. If the control frame is intended for a single DP slave, the L2 address between 0 and 125 must be entered here. If a group or several groups or all slaves are addressed by the control frame, the global address is entered for which the constant DPN_GROUP_ADR is defined in the dpn_user.h include file.

length

Here a value >= 1 must be entered.

slv_state

The control command of the call is entered here. The following values can be set:

Possible Entries	Command
DPN_FREEZE	FREEZE
DPN_UNFREEZE	UNFREEZE
DPN_SYNC	SYNC
DPN_UNSYNC	UNSYNC

user_data[0]

The group_identifier is entered in the user_data[0] structure element. The group_identifier selects the slave group (s) to be addressed. The group_identifier of a DP slave is specified when you create the DP database and the slave is informed of the identifier during parameter assignment.

A total of up to eight different groups can be created and selected using the individual bits of the group_identifier.

Example of a Group

If the value "3" in is entered in user_data[0], this selects groups "1" and "2" since bits 0 and 1 are set to “1” in the group_identifier.

Selecting All Slaves

To select all DP standard slaves, the constant DPN_SELECT_ALL must be entered in user_data[0] and the constant DPN_GROUP_ADR in the stat_nr structure element. The constants are defined in the dpn_user.h include file.

5.17.2 Return Parameters

ptr -> reference	unchanged
ptr -> stat_nr	unchanged
ptr -> length	unchanged
ptr **-> error_code**	**Error identifier**
ptr -> slv_state	unchanged
ptr -> **sys_state**	**Mode of the DP master**
ptr -> **sys_event**	**Event messages**
ptr -> user_data[0]	unchanged

error_code

The error_code structure element is identical to the return parameter of the function call.

The possible error identifiers are described in Section 4.4.

☞ **The entry of the constant “DPN_NO_ERROR“ in the error_code structure element simply means that the control commands were sent to the slaves. There is no confirmation that the job has been executed since the control frames are not acknowledged by the DP slaves.**

sys_state

The sys_state structure element contains the current mode of the DP master.

The possible modes are described in Section 4.8.3.

sys_event

The structure element sys_event identifies event messages during DP communication.

The possible event messages are described in Sections 3.4 and 4.8.4.

5.18 How a DP Application Logs Off

Call Function **unsigned short int dpn_reset (struct dpn_interface far * ptr)**

This call deletes the handle of the application and the resources assigned to the application are released.

Depending on the type and number of DP applications, various actions are executed:

Situation at the Time of the Call	Action
No further DP applications are logged on.	In this case, the entire DP communication is terminated. All allocated resources are released.
Other DP applications are active.	In this case, the output data of those DP slaves assigned to the DP application are set to 0.

☞ **This function must be called before a DP application is terminated.**

☞ **If a DP application is terminated due to an error without the dpn_reset() call, a safe system status can only be guaranteed when the DP application had previously set a watchdog time using the dpn_wd() function call.**

Reason:
Without the dpn_wd() call, the DP master would continue to output data to the DP slaves cyclically although the DP application is no longer active.

5.18.1 Call Parameters

ptr -> **reference**	**Handle of the DP application**
ptr -> stat_nr	irrelevant
ptr -> length	irrelevant
ptr -> error_code	irrelevant
ptr -> slv_state	irrelevant
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> user_data[ ]	irrelevant

reference The handle returned with the dpn_init() function call must be entered in the reference structure element.

5.18.2 Return Parameters

ptr -> reference	unchanged
ptr -> stat_nr	irrelevant
ptr -> length	irrelevant
ptr **-> error_code**	**Error identifier**
ptr -> slv_state	irrelevant
ptr -> sys_state	irrelevant
ptr -> sys_event	irrelevant
ptr -> user_data[ ]	irrelevant

error_code The error_code structure element is identical to the return parameter of the function call.

The possible error identifiers are described in Section 4.4. ❑

NOTES

6 Data Storage

This chapter describes the structure of the various data structures used by the call functions of the DP programming interface, as follows:

- ➢ Structure of the input and output data
- ➢ Structure of the diagnostic data
- ➢ Structure of the bus parameters
- ➢ Structure of the slave parameters
- ➢ Structure of the configuration data

6.1 Structure of the Input and Output Data

Contiguous Storage

The input and output data of a DP slave are stored contiguously starting from the user_data[0] structure element. The length structure element of the dpn_interface structure contains the number of valid bytes of the data area.

Output data sent to the DP slaves must be entered in the data area by the DP application before the function dpn_out_slv() or dpn_out_slv_m().

With functions that return data, the entry is made by the CP. These functions include dpn_in_slv(), dpn_in_slv_m() and dpn_read_slv(). The first two of these functions return the input data of one or more DP slaves while the dpn_read_slv() function returns the current output data from the **local** data buffer of the CP.

Order of the Data

The order of the data corresponds to the configured input/output ports of the DP slaves.

Analog values are an exception: see below

Example: The input ports of an ET 200B-16 DI station are read using the function dpn_in_slv(). The input value of port 0 is entered in the user_data[0] structure element, the input value of port 1 is entered in the user_data[1] structure element.

Exception: Analog Values

With older ET 200U stations that do not conform to the standard, the data are stored as follows:

- First, the analog values are stored in the order of the configured analog slots.
- Following these, the binary values are stored in the order of the configured digital modules.

Format of the Data Storage for Word Access

The following order is necessary for storing values in the word format:

First, the high byte (lower order address) of the word is entered followed by the low byte (higher order address).

☞ **This order does not correspond to the format of processors of the 8086 family!**

Exception: older ET 200 stations

On older ET 200U stations that do not conform to the standard (DP Siemens), data storage for word access corresponds to the format of the processors of the 8086 family:

First, the low byte (less significant address) and then the high byte (more significant address) of the word is entered.

6.2 Structure of the Diagnostic Data with Standard Slaves

Overview

The typical length of diagnostic data is between 6 and 32 bytes. The maximum possible length is 244 bytes. Diagnostic data have the structure described below.

Byte	Meaning
Byte 1	Station_status_1
Byte 2	Station_status_2
Byte 3	Station_status_3
Byte 4	Diag.Master_Add
Bytes 5,6	Ident_Number
Bytes 7-32	Ext_Diag_Data

Station_status_1

MSB							LSB
7	6	5	4	3	2	1	0

The individual bits have the following significance:

Bit 7: Diag.Master_Lock

The DP slave has already been assigned parameters by another master, in other words the local master does not currently have access to this slave.

Bit 6: Diag.Prm_Fault

This bit is set by the DP slave if the last parameter assignment frame was incorrect (for example wrong length, wrong ident number, invalid parameters).

Bit 5: Diag.Invalid_Slave_Response

This bit is set as soon as an implausible response is received from an addressed DP slave.

Bit 4: Diag.Not_Supported

This bit is set whenever a function is requested that is not supported by the particular slave (for example the SYNC mode is requested although the slave does not support this mode).

Bit 3: Diag.Ext_Diag

This bit is set by the DP slave. If the bit is set, there must be a diagnostic entry in the slave-specific diagnostic area (Ext_Diag_Data). If the bit is not set, there may be a status message in the slave-specific diagnostic area (Ext_Diag_Data). The meaning of the status message must be negotiated with each specific application.

Bit 2: Diag.Cfg_Fault

This bit is set when the configuration data last sent by the master do not match those determined by the DP slave, in other words when there is a configuration error.

Bit 1: Diag.Station_Not_Ready

This bit is set when the DP slave is not yet ready for productive data exchange.

Bit 0: Diag.Station_Non_Existent

This bit is set by the DP master if the DP slave cannot be reached via the bus. If this bit is set, the diagnostic bits contain the status of the last diagnostic message or the initial value. The DP slave always sets this bit to zero.

Station_status_2

MSB							LSB
7	6	5	4	3	2	1	0

The individual bits have the following significance:

Bit 7: Diag.Deactivated

This bit is set when the DP slave is marked as inactive in the local parameter data record and has been taken out of cyclic processing.

Bit 6: reserved

Bit 5: Diag.Sync_Mode

This bit is set by the DP slave as soon as it receives the sync control command.

Bit 4: Diag.Freeze_Mode

This bit is set by the DP slave as soon as it receives the freeze control command.

Bit 3: Diag.WD_On (Watchdog on)

This bit is set by the DP slave. If this bit is set to 1, the watchdog monitoring is activated on the DP slave.

Bit 2:

This bit is always set to 1 by the DP slave.

Bit 1: Diag.Stat_Diag (Static Diagnostics)

If the DP slave sets this bit, the DP master must fetch diagnostic information and continue to fetch it until the bit is cleared again. The DP slave, for example, sets this bit when it is not capable of providing valid user data.

Bit 0: Diag.Prm_Req

If the DP slave sets this bit, it requires a new parameter assignment and must be reconfigured. This bit remains set until the slave has had parameters assigned to it.

Note: If bit 1 and bit 0 are set, bit 0 has the higher priority.

Station_status_3

MSB							LSB
7	6	5	4	3	2	1	0

The individual bits have the following significance:

Bit 7: Diag.Ext_Diag_Overflow

If this bit is set, there is more diagnostic information than specified in Ext_Diag_Data. The DP slave sets this bit, for example, when there is more channel diagnostic data than the DP slave can enter in its send buffer. The DP master also sets this bit when the DP slave sends more diagnostic data than the DP master can accommodate in its diagnostic buffer.

Bit 0-6: reserved

Diag.Master_Add The address of the DP master that assigned parameters to this DP slave is entered in this byte. If the DP slave has not been assigned parameters by a DP master, the DP slave sets the address 255 in this byte.

Ident_Number The vendor's identifier is assigned for a DP slave type. This identifier can be used on the one hand for test purposes and on the other for precise identification of the slave.

Ext_Diag_Data The DP slave can enter its specific diagnostic data in this area. A field structure is necessary with a header byte both for the **device** and for the **identifier-related** diagnostics.

6.2.1 Device-related Diagnostics

Header Byte

MSB							LSB
7	6	5	4	3	2	1	0
Bit 7, bit 6 always 00		Block length in bytes including header byte 2 to header byte 63					

This field contains general diagnostic information such as overtemperature, undervoltage or overvoltage. The specific coding depends on the device. For further information, the Ident_Number is necessary.

6.2.2 Identifier-related Diagnostics

A bit is reserved for each identifier byte assigned during the configuration. The entries are always rounded up to byte boundaries and the non-configured bits are set to 0. If a bit is set, this means that there is diagnostic information in this I/O area.

Header Byte

MSB							LSB
7	6	5	4	3	2	1	0
Bit 7, bit 6 always 01		Block length in bytes including header byte 2 to header byte 63					

Bit Structure for Identifier-related Diagnostics

MSB							LSB
7	6	5	4	3	2	1	0

Bit 0: Identifier byte 0 contains diagnostic information
Bit 1: Identifier byte 1 has diagnostic information

"

Bit 7: Identifier byte 7 has diagnostic information

6.2.3 Channel-related Diagnostics

This field contains the diagnosed channels and the reason for the diagnostic information for each channel one after the other. The length of each entry is 3 bytes.

Byte 1: Identifier Number

MSB LSB

7	6	5	4	3	2	1	0
Bit 7, bit 6 always 10		Identifier number 0 to 63					

Byte 2: Channel Number

MSB LSB

7	6	5	4	3	2	1	0
Input/output 00 reserved 01 input 10 output 11 input/output.		Channel number 0 to 63					

With identifier bytes that contain both inputs and outputs, bits 7 and 6 of the channel number contain the direction of the diagnosed channel.

Byte 3: Type of Diagnostics

MSB LSB

7	6	5	4	3	2	1	0
Channel type 000 reserved 001 bit 010 2 bits 011 4 bits 100 byte 101 word 110 2 words 111 reserved			Error type				

Error type:

0	reserved
1	short-circuit
2	undervoltage
3	overvoltage
4	overload
5	overtemperature
6	line break
7	upper limit value exceeded
8	lower limit value exceeded
9	error
10-15	reserved
16-31	vendor-specific

6.2.4 Example: Structure of the Diagnostic Information

	MSB 7	6	5	4	3	2	1	LSB 0
Device-related diagnostics:	0	0	0	0	0	1	0	0
The meaning of the bits is specified by the vendor	Device-specific diagnostic field with length 3							
Identifier-related diagnostics:	0	1	0	0	0	1	0	1
ID number 0 with diag. info.	0	0	0	0	0	0	0	1
ID number 1 with diag. info.	0	0	0	1	0	0	0	0
ID number 18 with diag. info.	0	0	0	0	0	1	0	0
	0	0	0	0	0	0	0	0
Channel-related diagnostics: ID number 0	1	0	0	0	0	0	0	0
Channel 2	0	0	0	0	0	0	1	0
Overload, channel organized in bits	0	0	1	0	0	1	0	0
ID number 12	1	0	0	0	1	1	0	0
Channel 6	0	0	0	0	0	1	1	0
Upper limit value exceeded, channel organized in words	1	0	1	0	0	1	1	1

6.3 Diagnostic Data of Non-standard Slaves

In various older versions of the ET 200U, ET 200K and ET 200B station types, the structure of the diagnostic information is different from today's standard. The structure of the data with these types is explained below.

6.3.1 Diagnostic Data of the ET 200U

General Structure

Byte 1	Station diagnostic information
Byte 2	Station status
Byte 3-6	Module diagnostic information

Structure of the Station Diagnostic Information

MSB							LSB
7	6	5	4	3	2	1	0

The individual bits have the following significance:

Bit 0: irrelevant
Bit 1: Station cannot be controlled
Bit 2: Parameter assignment error (module identifier)
Bit 3: Single error (module removed)
Bit 4: No load voltage
Bit 5: Output incorrectly activated
Bit 6: irrelevant
Bit 7: Extended diagnostics activated

Structure of the Station Status

MSB							LSB
7	6	5	4	3	2	1	0

The individual bits have the following significance:

Bit 0-2: irrelevant
Bit 3: Watchdog monitoring activated
Bit 4: S5-100 slow mode set
Bit 5: No further diagnostic message is generated
Bit 6: irrelevant
Bit 7: No new data are written to the outputs

Structure of the Module Diagnostic Information

MSB							LSB
7	6	5	4	3	2	1	0

The individual bits have the following assignment:

Byte 3:

Bit 0: Module in slot 0
Bit 1: Module in slot 1
Bit 2: Module in slot 2

....(etc.) ....

Byte 6:

Bit 7: Module in slot 31

If a bit is set to 1, an error has occurred on the corresponding module.

6.3.2 Diagnostic Data of the ET 200K/B

General Structure

Byte 1	Status byte 1
Byte 2	Status byte 2
Byte 3-6	Diagnostic port A-D

Structure of Status Byte 1

MSB							LSB
7	6	5	4	3	2	1	0

The individual bits have the following significance:

Bit 0: RESET was triggered
Bit 1: Watchdog timer expired
Bit 2: Diagnostic error
Bit 3: Error with EEPROM write command
Bit 4: Ports were frozen
Bit 5: Output to ports is disabled
Bit 6-7: irrelevant

Structure of Status Byte 2

MSB							LSB
7	6	5	4	3	2	1	0

The individual bits have the following significance:

Bit 0: irrelevant
Bit 1=0: Watchdog deactivated
Bit 2: irrelevant
Bit 3: irrelevant
Bit 4-6: irrelevant
Bit 7: EEPROM connected

Structure of the Diagnostic Ports (A-D) in ET 200K Stations

MSB							LSB
7	6	5	4	3	2	1	0

The individual bits have the following significance:

Byte 3 (Diagnostic Port A):

Bit 0: Input A0 error
Bit 1: Input A1 error

.....

Bit 7: Input A7 error

Byte 4 (Diagnostic Port B):

Bit 0: Input B0 error
Bit 1: Input B1 error

.....

Bit 7: Input B7 error

Byte 5 (Diagnostic Port C):

Bit 0: Input C0 error
Bit 1: Input C1 error

.....

Bit 7: Input C7 error

Byte 6 (Diagnostic Port D):

Bit 0: Input/output D0 error
Bit 1: Input/output D1 error

.....

Bit 7: Input/output D7 error

Structure of the Diagnostic Ports (A-D) in ET 200B Stations

Of the ET 200B devices, only type ET 200B 16DQ has diagnostic capability. The diagnostic ports are only relevant with this type.

MSB							LSB
7	6	5	4	3	2	1	0

Byte 3 (Diagnostic Channel Group 0):

The individual bits have the following significance:

Bit 0=0: Overload, output short-circuit
Bit 1-6: irrelevant
Bit 7=0: Fuse defective, no load voltage

Byte 4 (Diagnostic Channel Group 1):

Bit 0=0: Overload, output short-circuit
Bit 1-6: irrelevant
Bit 7=0: Fuse defective, no load voltage

6.4 Structure of the Bus Parameters

The bus parameters have the following structure:

Name of the Parameter	Type (Intel format)
Reserved	unsigned16
FDL_Add	unsigned8
Baudrate	unsigned8
T_{SL} *)	unsigned16
minT_{SDR} *)	unsigned16
maxT_{SDR} *)	unsigned16
T_{QUI} *)	unsigned8
T_{SET} *)	unsigned8
T_{TR} *)	unsigned32
G *)	unsigned8
HSA *)	unsigned8
max_retry_limit *)	unsigned8
Bp_Flag	unsigned8
Min_Slave_Interval	unsigned16
Poll_Timeout	unsigned16
Data_Control_Time	unsigned16

*) These parameters are described in DIN 19245 Part 1.

FDL_Add L2 address of the DP master.

Baudrate Code number for the transmission rate

Code Number	Transmission Rate
0	9.6 Kbps
1	19.2 Kbps
2	93.75 Kbps
3	187.5 Kbps
4	500 Kbps
5	375 Kbps
6	750 Kbps
7	1.5 Mbps
8	3 Mbps
9	6 Mbps
10	12 Mbps

Bp_Flag

MSB							LSB
7	6	5	4	3	2	1	0

Bit 7 (Error_Action_Flag)

Bit 7=0: no mode change if error occurs

Bit 7=1: mode change if error occurs (AUTOCLEAR)

Bit 0-6 reserved

Min_Slave_Interval This time is important when sending control frames and in the gap between polling cycles. For a detailed description of the parameter refer to Section 3.7.4.

Time unit: 5 ms

Poll_Timeout This time is used to monitor the communication between the DP master Class 1 and a DP master Class 2 (DP diagnostic unit). For a detailed description of the polling timeout refer to Section 3.7.3.

Time unit: 50 ms

Data_Control_Time This time is required for the AUTOCLEAR function. A detailed description of the data control time can be found in Section 3.7.2.

Time unit: 50 ms

6.5 Structure of the Slave Parameters

Overview

The slave parameters consist of the following components

- Sl_Flag, slave type, octet string
- Parameter assignment data
- Configuration data
- Add-Tab list
- Slave-User-Data

The components listed above can be read out with the dpn_read_slv_par() call function. This function is described in Chapter 5.

6.5.1 Sl_Flag, Slave Type, Octet String

These slave parameter components have the following structure:

Name	Type
Sl_Flag	unsigned8
Slave type	unsigned8
Octet 1 (reserved)	unsigned8
....	....
Octet 12 (reserved)	unsigned8

Sl_Flag

This parameter contains slave-related flags.

MSB LSB

7	6	5	4	3	2	1	0

Bit 7 (Active)

Bit 7 = 0: DP slave is not activated.

Bit 7 = 1: DP slave is activated.

Bit 6 (New_Prm)

Bit 6 = 0: DP slave receives user data.

Bit 6 = 1: DP slave receives new parameter assignment data.

Bits 0-5

These bits are reserved

Slave Type

This parameter contains a vendor-specific type identifier for the slave unit.

Range: 0 to 2^8-1

0	DP standard slave
1 to 15	reserved
16 to 255	vendor-specific

6.5.2 Parameter Assignment Data

The parameter assignment data consist of bus-specific data and DP slave-specific data.

Type: Octet string

Length: Ideally 7 to 32

Octet Number	Meaning
Octet 1	Station_status
Octet 2	WD_Fact_1
Octet 3	WD_Fact_2
Octet 4	Min. Station Delay Responder
Octet 5-6	Ident_Number
Octet 7	Group_Ident
Octet 8 to 32	User_Prm_Data

Station_status

Structure of the station_status:

MSB							LSB
7	6	5	4	3	2	1	0

The individual bits have the following significance:

Bits 7 and 6 Lock_Req and Unlock_Req

Bit 7	Bit 6	Meaning of the Bit Combinations
0	0	The min T_{SDR} is overwritten when parameters are assigned. All other parameters remain unchanged.
0	1	The DP slave is enabled for other masters.
1	0	The DP slave is disabled for other masters, all parameters are adopted (exception: min T_{SDR} = 0).
1	1	The DP slave is enabled for other masters.

Bit 5 Sync_Req

This bit indicates to the slave that it must operate in the Sync mode as soon as the command is transferred with the dpn_global_crtl() function. If the CP slave does not support the Sync command, it sets the bit Diag.Not_Supported in the diagnostic information.

Bit 4 Freeze_Req

This bit indicates to a DP slave that it must operate in the freeze mode as soon as the command is transferred with the dpn_global_ctrl() function. If the DP slave does not support the freeze command, it sets the bit Diag.Not_Supported in the diagnostic information.

Bit 3 Watchdog

If this bit is set to 0, the watchdog monitoring is deactivated. If the bit is set, the watchdog monitoring is activated on the DP slave.

Bits 2, 1, 0

These bits are reserved for future expansions. If no expansions are implemented on a DP slave and one of these bits is set, the bit Diag.Not_Supported is set in the diagnostic information.

WD_Fact_1
WD_Fact_2

These two bytes contain factors for setting the watchdog time (T_{WD}). The watchdog ensures that if this time expires after a DP master has failed, the outputs are set to a safe state.

T_{WD} [ms] = 10 ms * WD_Fact_1 * WD_Fact_2

Min. Station Delay Responder

This is the time that the DP slave must wait before it is allowed to send its response frames to the DP master.

Unit: bit times

Ident_Number

This number is assigned by the vendor. The DP slave only accepts parameter assignment frames when the Ident_Number transferred with the parameter assignment data matches its own Ident_Number.

Group_Ident

With this parameter, a group can be formed for the dpn_glb_ctrl() function. Each bit represents a group. The Group_Ident is only accepted when the Lock_Req bit is set.

User_Prm_Data

These bytes can be used for parameters specific to the DP slave (for example diagnostic filter, controller parameters). The meaning and range of values are specified by the vendor.

6.5.3 Configuration Data

The configuration data contain the size of the input and output data areas and information about the data consistency.

Length

The length of the configuration data is ideally 1 to 32. If necessary however, up to 244 bytes are possible.

Structure

Input and output areas can be grouped together and described by an identifier byte.

Structure of the identifier byte

MSB LSB

7	6	5	4	3	2	1	0

Bits 0-3 Length of the data

00 1 byte or 1 word

...

15 16 bytes or 16 words

Bits 5,4 Input/output

00 Special identifier formats
01 Input
10 Output
11 Input/output

Bit 6 Length

0 Byte (byte structure)
1 Word (word structure)

Bit 7 (Consistency)

0 Byte or word
1 Total length

6.5.4 Special Identifier Formats

Special identifier formats allow the configuration to be extended by increasing the flexibility.

Structure of the Identifier

MSB							LSB
7	6	5	4	3	2	1	0

Bits 0-3 Length of the vendor-specific data

These bits contain the length of the vendor-specific data.

Bits 4,5

These bits are always set to 0.

Bits 7,6 Input/output

00 Empty
01 1 length byte for inputs follows.
10 1 length byte for outputs follows.
11 1 length byte for outputs and
1 length byte for inputs follow.

Structure of the Length Byte

MSB							LSB
7	6	5	4	3	2	1	0

Bits 0-5 length of the inputs/outputs

00 1 byte or 1 word
...
63 64 bytes or 64 words

Bit 6 Length

0 Byte (byte structure)
1 Word (word structure)

Bit 7 Consistency

0 Byte or word
1 Total length

Example of a Special Identifier Format

Oct.1	1	1	0	0	0	0	1	1	Output/input, 3 bytes of vendor-specific data
Oct.2	1	1	0	0	1	1	1	1	Consistency, output, 16 words
Oct.3	1	1	0	0	0	1	1	1	Consistency, input, 8 words
Oct.4									Vendor-
Oct.5									specific
Oct.6									data

❑

NOTES

7 Creating DOS Applications

Libraries and Include Files Required

To create DP applications, you require the following files from the installation diskette:

Files on the Installation Diskette	Meaning
DP library	Library of the DP programming interface.
DP include file	DP-specific declarations and constants for the DP programming interface.
SCI library	Library with functions for accessing the driver of the CP.

The DP library and the SCI library must be linked to the DP application. The DP include file provides the required declarations and constants for the DP application.

Overview of the Components

The following table shows the compiler versions for which the DP library is available. The corresponding columns of the table show which files are required for creating the DP application. For example, to create a Borland DOS application, you must link the libraries lddpnbc.lib and ldscitc.lib to the DP application.

Compiler Files	Microsoft MSVC 1.x	Borland C++ V 4.x	Comment
DP library	lddpnmsc.lib	lddpnbc.lib	DP library for DOS in the large model.
DP include file	dpn_user.h	dpn_user.h	DP-specific declarations.
SCI library	ldscimsc.lib	ldscitc.lib	These libraries must be linked to the DP application.

The ldscimsc.lib library was created with Microsoft Visual C++ Version 1.52 and the ldscitc.lib library was created with TurboC++ Version 1.0.

Note

When compiling the DP application, the define command DPN_DOS must be activated (for example using a compiler switch). This define command is required by the dpn_user.h include file and **must be made available by the user.**

7.1 Environment Under DOS

Overview

This chapter provides you with information about using a DP application under DOS.

The two modes

- ➢ single board mode
- ➢ multiboard mode

are possible.

Single Board Mode

Fig. 7.1 illustrates the single board mode of a DP application under DOS. The DP application accesses **one** CP.

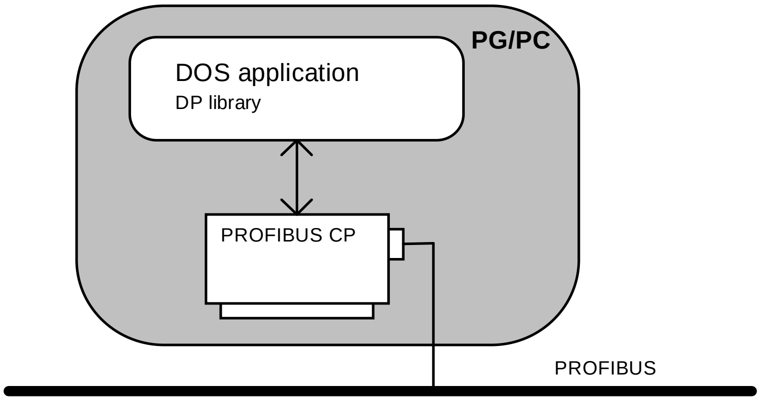


Fig. 7.1: Single Board Mode under DOS

Multiboard mode

You can install **more than one** PROFIBUS CP in one computer. In the multiboard mode, a DP application accesses more than one CP as shown in Fig. 7.2.

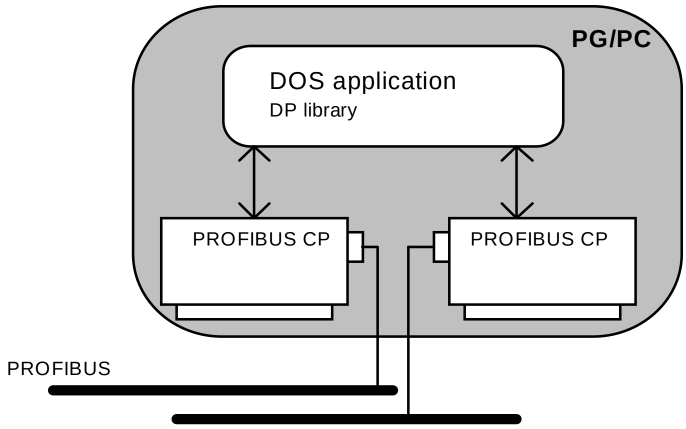


Fig. 7.2: Multiboard Mode with Two Modules

Requirements for the Multiboard Mode

In the multiboard mode, the PROFIBUS modules must be connected to different buses. Each of these CPs acts as the DP master controlling the DP slaves on its bus.

☞ **Operation of more than one DP master in the same PG/PC and on the same bus leads to conflicts and is therefore not permitted.**

Number of PROFIBUS Modules in the Multiboard Mode

The DP library supports up to two CP modules.

7.2 Logging On a DP Application

General

A DOS application must log on using the function dpn_init() for **every** CP module with which it wants to communicate. This function must be called **before all other** DP functions.

The function call specifies the following characteristics of the DP application and makes them known to the CP (for a detailed description, see Chapter 5).

- ➢ The number of the PROFIBUS module with which the DP application communicates.
- ➢ The type and environment of the DP application.
- ➢ The rights of access to the DP slaves.

If the dpn_init() is successful, a reference (handle) to the selected CP is returned. The reference must be used with all further function calls to this CP in the reference element of the dpn.interface structure.

Call Parameters

Parameter	Possible Entries	Comment
reference.board_select	1 - 2	Number of the CP
reference.access	(DPN_SYS_CENTRAL) | (DPN_ROLE_CENTRAL) (DPN_SYS_NOT_CENTRAL) | (DPN_ROLE_NOT_CENTRAL)	Type and environment of the DP application (see also Chapter 5)
user_data[n] where n=0 to 125	DPN_SLV_WRITE_READ DPN_SLV_READ DPN_SLV_NO_ACCESS	Rights of access to DP slaves (see also Chapter 5)

reference.access for dpn_init() under DOS

Using this parameter, a DP application identifies itself to the DP master when using the dpn_init() call. According to the table above, the two following entries are permitted under DOS:

- ➢ (DPN_SYS_CENTRAL) **|** (DPN_ROLE_CENTRAL)
- ➢ (DPN_SYS_NOT_CENTRAL) **|** (DPN_ROLE_NOT_CENTRAL)

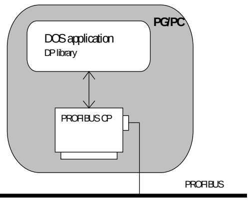
The following table shows the different responses of the DP master to these entries.

	(DPN_SYS_CENTRAL) | (DPN_ROLE_CENTRAL)	(DPN_SYS_NOT_CENTRAL) | (DPN_ROLE_NOT_CENTRAL)
Setting the mode of the DP master	The mode of the DP master must be set by the DP application using the dpn_set_mode() function call. The mode can be changed at any time.	Following the dpn_init() call, the DP master automatically changes to the OPERATE mode.
Available DP functions	All the DP functions of the DP programming interface can be used.	The following DP function calls are not permitted: dpn_load_bus_par() dpn_set_mode() dpn_global_ctrl()
Miscellaneous	If AUTOCLEAR was configured, the DP master automatically changes to the CLEAR mode if an error occurs.	The AUTOCLEAR function is disabled.

7.2.1 Examples of Logging On a DP Application

The following examples show the basic procedure for logging on in the single board and multiboard modes. The examples have been simplified to make them easier to understand. For example, the return value is not checked which it should always be in a real DP application.

Example: Single Board Mode



One CP is installed in a computer. Three DP slaves are connected to the bus and have the L2 addresses 3, 4 and 7. DP slave no. 3 has only input ports, DP slave no. 4 has only output ports and DP slave no. 7 has both input and output ports.

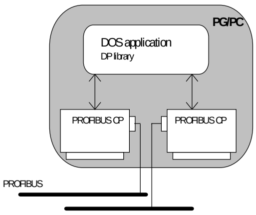
After the dpn_init() function call, the DP master will change automatically to the OPERATE mode. For this reason, the identifier (DPN_SYS_NOT_CENTRAL | DPN_ROLE_NOT_CENTRAL) is entered in the reference.access structure.

The dpn_ptr pointer in the example points to the dpn_interface structure.

The initialization call could appear as follows:

```
dpn_ptr -> reference.board_select =  1;
dpn_ptr -> reference.access =        (DPN_SYS_NOT_CENTRAL) |
                                     (DPN_ROLE_NOT_CENTRAL);
dpn_ptr -> length =          8; // Index 0 to 7
dpn_ptr -> user_data[0] =     DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[1] =     DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[2] =     DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[3] =     DPN_SLV_READ;
dpn_ptr -> user_data[4] =     DPN_SLV_WRITE_READ;
dpn_ptr -> user_data[5] =     DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[6] =     DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[7] =     DPN_SLV_WRITE_READ;
error = dpn_init (dpn_ptr);
```

Example: Multiboard Mode



Two CPs are installed in one computer. Each CP is connected to a separate bus segment.

The arrangement of the slaves on bus 1 is the same as in the example above for the single mode board. The DP master for bus 1 changes automatically to the OPERATE mode after the dpn_init() function call.

One DP slave with L2 address 3 is connected to bus 2. The DP slave has only output ports. In contrast to bus no. 1, the DP application will set the OPERATE mode. For this reason, the identifier (DPN_SYS_CENTRAL | DPN_ROLE_NOT_CENTRAL) is entered in the reference.access structure element.

The dpn1_ptr pointer points to the dpn_interface structure of bus 1. The dpn2_ptr pointer points to the dpn_interface structure of bus no. 2.

The initialization calls of the DP application could appear as follows:

```
dpn1_ptr -> reference.board_select =        1;
dpn1_ptr -> reference.access =      (DPN_SYS_NOT_CENTRAL)|
                                    (DPN_ROLE_NOT_CENTRAL);
dpn1_ptr -> length =                8; // Index 0 to 7
dpn1_ptr -> user_data[0] =          DPN_SLV_NO_ACCESS;
dpn1_ptr -> user_data[1] =          DPN_SLV_NO_ACCESS;
dpn1_ptr -> user_data[2] =          DPN_SLV_NO_ACCESS;
dpn1_ptr -> user_data[3] =          DPN_SLV_READ;
dpn1_ptr -> user_data[4] =          DPN_SLV_WRITE_READ;
dpn1_ptr -> user_data[5] =          DPN_SLV_NO_ACCESS;
dpn1_ptr -> user_data[6] =          DPN_SLV_NO_ACCESS;
dpn1_ptr -> user_data[7] =          DPN_SLV_WRITE_READ;
error = dpn_init (dpn1_ptr);

dpn2_ptr -> reference.board_select =        2;
dpn2_ptr -> reference.access =      (DPN_SYS_CENTRAL) |
                                    (DPN_ROLE_CENTRAL);
dpn2_ptr -> length =                4; // Index 0 to 3
dpn2_ptr -> user_data[0] =          DPN_SLV_NO_ACCESS;
dpn2_ptr -> user_data[1] =          DPN_SLV_NO_ACCESS;
dpn2_ptr -> user_data[2] =          DPN_SLV_NO_ACCESS;
dpn2_ptr -> user_data[3] =          DPN_SLV_WRITE_READ;
error = dpn_init (dpn2_ptr);
// Here, set the mode of bus no. 2 using the
// function calls dpn_set_mode()
// and dpn_get_mode(). See Chapter 5 of the DP description
```

7.3 Porting DP Applications of the TF-5412 Product

The present DP programming interface continues to support the interface functions of the TF-5412/MS-DOS, Windows product.

☞ **The old interface calls of the TF 5412/MS-DOS, Windows product must only be used for the slaves listed in the corresponding manual.**

☞ **When DP applications are created new, only the new DP call functions (described in Chapter 5) can be used.**

☞ **Simultaneous use of “new” and “old” (TF 5412/ MS-DOS, Windows) function calls in a DP application is not permitted.**

☞ **The AUTOCLEAR function that can be set with the COML DP configuration tool has no effect with older applications.**

Procedure

Porting involves the following two steps:

➢ New translation of the application with one of the compilers listed in Chapter 7. Here, the “old” dp_user.h include file must be replaced by the "new" dp_user.h include file from the installation diskette (do not confuse this with the dpn_user.h include file).

➢ The libraries lddpnmsc.lib and ldscimsc.lib (Microsoft applications) or lddpnbc.lib and ldscitc.lib (Borland applications) from the installation diskette must be linked.

Restrictions

When porting old applications, only the range of functions described in the TF 5412 /MSDOS, Windows manual (Chapter 8) is supported. This means, among other things:

➢ Access possible to only **one** CP

➢ Maximum number of DP slaves = 31

➢ **No** multitasking mode permitted

➢ Operation possible **only under DOS**

Differences Compared with the CP 5412 (A1)

- ➢ Owing to the different hardware and software architecture, the dynamic response can change compared with the CP 5412 (A1).
- ➢ The ET 200 initialization function dp_init() expects the first transferred parameter to be a pointer to the path and name of a database created with COM ET 200. Instead of this pointer, a ZERO pointer can now be transferred as an **option** if a database created with the new COML DP configuration tool is being used. In contrast to the CP 5412 (A1), a database created with COML DP is not loaded by the DP application but along with the firmware during the startup phase. Of course, the pointer to the database can continue to be transferred with the dp_init() call if no database is being used that was created with COML DP.
- ➢ If an error occurs, the return parameters (error IDs) of the function calls may be different from those of the old ET 200 DP library.
- ➢ The granularity of the watchdog function with the dp_wd() of the old DP library (CP 5412 (A1)) is 320 ms. With the CP 5412 (A2), the granularity of this function is now 400 ms.
- ➢ On the CP 5412 (A1), the dp_dia_s() diagnostic function does not return diagnostic data when communication to the DP slave is interrupted. On the CP 5412 (A2), the last received diagnostic data are returned even when the communication between the DP master and DP slave is interrupted.
- ➢ If the watchdog time expires due to an error in the DP application after calling the dp_wd() function, a further dp_wd() call must be made to allow data exchange with the DP slaves to be resumed. ❑

NOTES

8 Creating Windows Applications

Supported Windows Versions

The Microsoft operating systems Windows 3.x in enhanced mode, Windows 95 and Windows NT are supported.

Libraries and Include Files Required

The DP library is made available under Windows as a DLL (Dynamic Link Library).

The DP include file makes the necessary declarations and constants available to the DP application.

Windows 3.x

To create DP applications under Windows, you require the following files from the installation diskette.

Files of the Installation Diskette	File Name	Created with	Meaning
DP Library	lwdpnbc.dll	BC++ 4.0	BC++ 4.0 DP functions in a DLL
DP Import Library	lwdpnbc.lib	BC++ 4.0	BC++ 4.0 DP function prototypes
DP Include File	dpn_user.h		DP-specific declarations and constants

Windows 95 and Windows NT

To create DP applications under Windows 95 and Windows NT, you require the following files from the installation diskette.

Files of the Installation Diskette	File Name	Crated with	Meaning
DP Library	dplib.dll	See product information	DP functions in a DLL
DP Import Library	dplib.lib	See product information	DP function prototypes
DP Include File	dpn_user.h		DP-specific declarations and constants

Constants

When compiling the DP application, the define statement DPN_WIN must be activated (for example with compiler switches). This define statement is required by the dpn_user.h include file and **must be provided by the user**.

8.1 Environment Under Windows

Overview

This section provides you with information about using a DP application under Windows.

A distinction must be made between the modes:

- ➢ Single board/multiboard mode
- ➢ Single user/multi-user mode

The possible combinations are explained in the following sections.

Single Board Mode/Single User Mode

Fig. 8.1 illustrates the situation for the single board and single user mode. Here only **one** DP Windows application accesses **one** PROFIBUS CP.

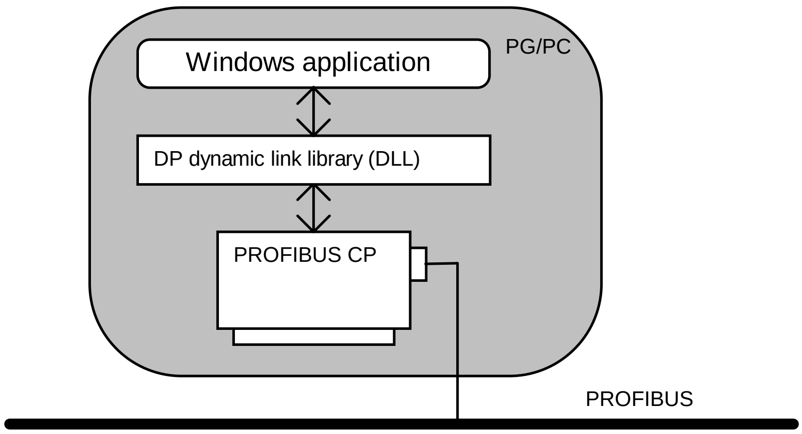


Fig. 8.1: Single Board/Single User Mode

Single Board Mode/Multi-User Mode

In this case, **several** DP Windows applications access **one** common PROFIBUS CP.

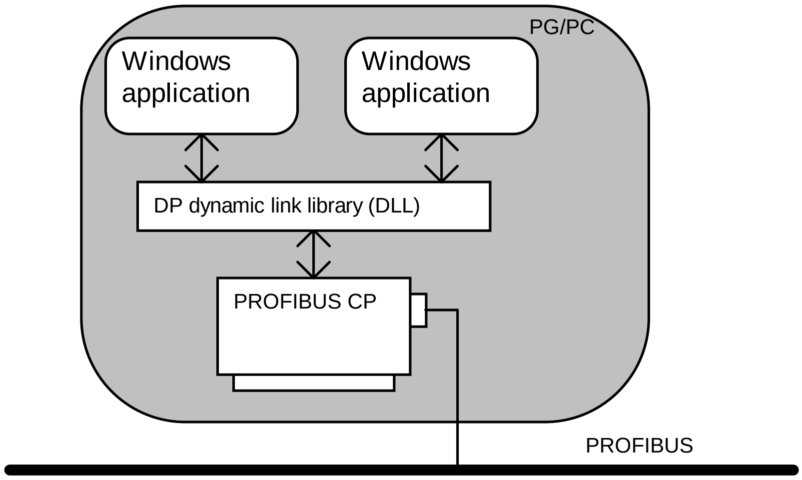


Fig. 8.2: Single Board/Multi-User Mode

Multiboard/Single User Mode

More than one PROFIBUS CP can be installed in one computer. In the multiboard/single user mode, **one** DP Windows application accesses **more than one** PROFIBUS CP as shown in Fig. 8.3.

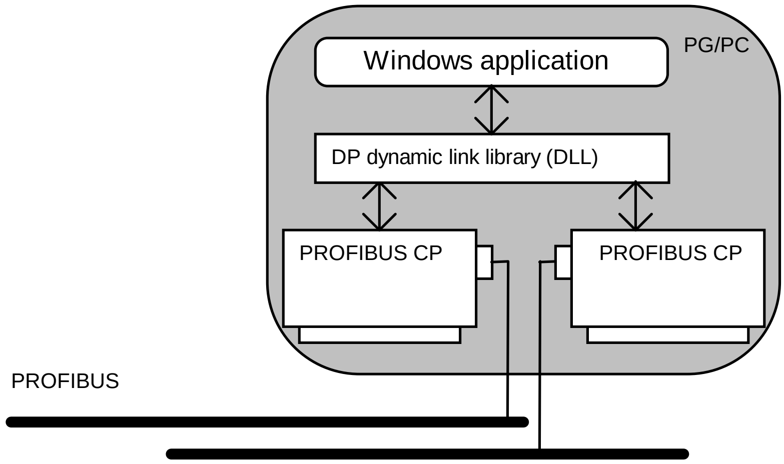


Fig. 8.3: Multiboard/Single User Mode

Multiboard/Multi-User Mode

More than one PROFIBUS CP can be installed in one computer. In the multiboard/multi-user mode, **more than one** DP Windows application accesses **more than one** PROFIBUS CP as shown in Fig. 8.4.

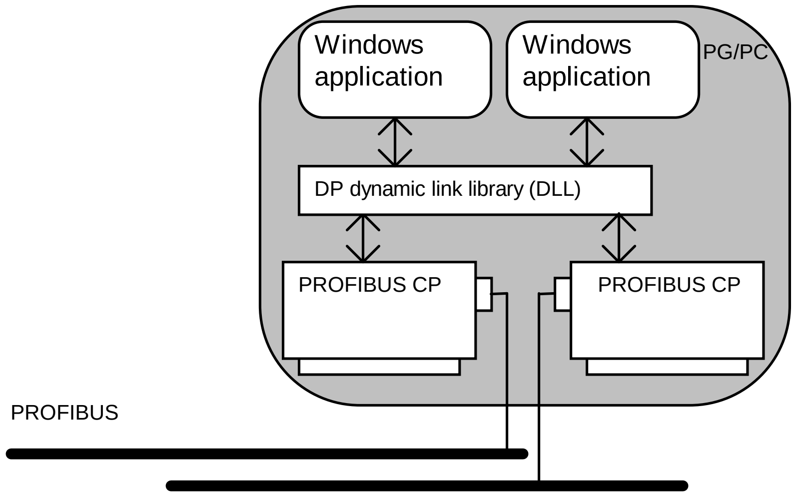


Fig. 8.4: Multiboard/Multi-User Mode

Requirements for the Multiboard Mode

In the multiboard mode, the CPs must be connected to different buses. Each of these CPs controls the DP slaves of the bus as the DP master.

☞ **Operating more than one DP master in the same PC/PG and on the same bus leads to conflicts and is therefore not permitted.**

Number of PROFIBUS Modules in the Multiboard Mode

The DP-DLL supports a maximum of two PROFIBUS modules.

Number of DP Windows Applications

The DP-DLL supports a maximum of four DP Windows applications.

☞ A maximum of **two CPs** and **four applications** can be supported at the same time. Remember that the logging on of **one DP application** at **two CPs** (**two log on calls** are required) is handled like the logging on of **two applications**, in other words after this log on, **two further applications** can also log on.

Example 1:

The DP applications 1, 2 and 3 log on for access to CP 1. DP application 4 logs on for access to CP 2. All the logons (provided they are not contradictory) are acknowledged positively.

A further DP application 5 wants to log on at module 2. The log on is acknowledged negatively since four applications are already logged on.

Example 2:

The DP application 1 logs on at CPs 1 and 2. DP application 2 also logs on at CPs 1 and 2. Assuming that all four logons were acknowledged positively, DP application 3 then attempts to log on at CP 1.

This log on is acknowledged negatively since the log on of DP application 1 (2) on CPs 1 and 2 is treated as a log on of 2 DP applications. This means that at the time when DP application 3 attempts to log on, a total of four DP applications are already logged on.

8.2 Logging On a DP Windows Application

General

A Windows application must log on using the dpn_init() function at **every** PROFIBUS module with which it wants to communicate. This function must be called **before all other** DP functions.

Along with the function call, the following characteristics of the DP application are specified and made known to the CP (for a detailed description see Chapter 5):

- The number of the CP
- Type and environment of the DP application
- Rights of access to the DP slaves

If successful, the dpn_init() function returns a reference (handle) to the CP. The reference must be entered in all further function calls to this DP in the reference element of the dpn_interface structure.

Call Parameters

Parameter	Possible Entries	Comment
reference.board_ select	1 - 2	Number of the CP
reference.access	(DPN_SYS_CENTRAL) | (DPN_ROLE_CENTRAL) (DPN_SYS_CENTRAL) | (DPN_ROLE_NOT_CENTRAL) (DPN_SYS_NOT_CENTRAL) | (DPN_ROLE_NOT_CENTRAL)	Type and environment of the DP application
user_data[n] where n=0 to 125	DPN_SLV_WRITE_READ DPN_SLV_READ DPN_SLV_NO_ACCESS	Right of access to DP slaves

reference.access with dpn_init() under Windows

Using this structure element, a DP application must identify itself to the DP programming interface when using the dpn_init() call. According to the table above. the following entries are permitted under Windows:

- (DPN_SYS_CENTRAL) | (DPN_ROLE_CENTRAL)
- (DPN_SYS_NOT_CENTRAL) | (DPN_ROLE_NOT_CENTRAL)
- (DPN_SYS_CENTRAL) | (DPN_ROLE_NOT_CENTRAL)

The table below, shows the different responses of the DP-DLL to the possible entries.

Setting the Type of the Application	Meaning	Available DP Functions
(DPN_SYS_CENTRAL) | (DPN_ROLE_CENTRAL)	The DP application is a central application. The mode of the DP master must be set by the DP application using the dpn_set_mode() function call. The mode can be changed at any time.	All the DP functions of the DP-DLL can be used.
(DPN_SYS_CENTRAL) | (DPN_ROLE_NOT_ CENTRAL)	The DP application is not a central application. Another DP application exists. This other DP application is a central application. It must set the mode of the DP master.	The following DP function calls are not permitted: dpn_load_bus_par() dpn_set_mode() dpn_ global_ctrl().
(DPN_SYS_NOT_ CENTRAL) | (DPN_ROLE_NOT_ CENTRAL)	The DP application is not a central application. The DP master changes automatically to the OPERATE mode after the dpn_init() call since there is no central application in the system.	The following DP function calls are not permitted: dpn_load_bus_par() dpn_set_mode() dpn_global_ctrl() <u>AUTOCLEAR:</u> The AUTOCLEAR function (see Section 3.7.6) has no effect with the DPN_SYS_NOT_ CENTRAL setting.

Special Case: Multi-User/Single Board Mode

If more than one DP Windows application logs on at the **same** CP, remember the following rules:

Rule 1 Only one of the DP applications can log on as the central DP application (DPN_ROLE_CENTRAL).

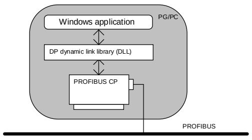
Rule 2 All DP applications must enter the same value as the DP application environment (either DPN_SYS_CENTRAL or DPN_SYS_NOT_CENTRAL).

Rule 3 No more than one DP application can send output data to any one DP slave. For this reason, when the access rights to a particular slave x are assigned, the identifier DPN_SLV_ WRITE_READ (write output data/read input data) must only be assigned by **one** DP Windows application.

8.2.1 Examples of Logging On Under Windows

The following examples show the basic log on procedure for DP applications in the single board and multiboard mode. To make them clearer, the examples have been simplified. There is for example no check of the return value which should always be the case in a real application.

Example 1: Single Board-/ Single User Mode



One CP is installed in a computer. Three DP slaves are connected to the bus and have the L2 addresses 3, 4 and 7. DP slave no. 3 has only input ports, DP slave no. 4 has only output ports and DP slave no. 7 has both input and output ports.

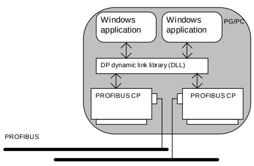
After the dpn_init() function call, the DP master will change automatically to the OPERATE mode. For this reason, the identifier (DPN_SYS_NOT_CENTRAL | DPN_ROLE_NOT_CENTRAL) is entered in the reference.access structure.

The dpn_ptr pointer in the example points to the dpn_interface structure.

The initialization call could appear as follows:

```
dpn_ptr -> reference.board_select =  1;
dpn_ptr -> reference.access =        (DPN_SYS_NOT_CENTRAL) |
                                     (DPN_ROLE_NOT_CENTRAL);
dpn_ptr -> length =                  8; // Index 0 to 7
dpn_ptr -> user_data[0] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[1] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[2] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[3] =            DPN_SLV_READ;
dpn_ptr -> user_data[4] =            DPN_SLV_WRITE_READ;
dpn_ptr -> user_data[5] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[6] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[7] =            DPN_SLV_WRITE_READ;
error = dpn_init (dpn_ptr);
```

Example 2: Multiboard-/Multi-User Mode



Two CPs are installed in one computer. Each CP is connected to a separate bus segment.

The arrangement of the slaves on bus 1 is the same as in the example above for the single mode board. The DP master for bus 1 changes automatically to the OPERATE mode after the dpn_init() function call.

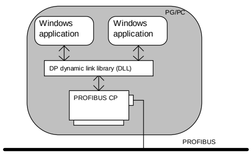
One DP slave with L2 address 3 is connected to bus 2. The DP slave has only output ports. In contrast to bus no. 1, the DP application will set the OPERATE mode. For this reason, the identifier (DPN_SYS_CENTRAL) is entered in the reference.access structure element.

The dpn1_ptr pointer points to the dpn_interface structure of bus 1. The dpn2_ptr pointer points to the dpn_interface structure of bus no. 2.

The initialization calls of the DP application could appear as follows:

```
dpn1_ptr -> reference.board_select = 1;
dpn1_ptr -> reference.access =       (DPN_SYS_NOT_CENTRAL) |
                                     (DPN_ROLE_NOT_CENTRAL);
dpn1_ptr -> length =                 8; // Index 0 to 7
dpn1_ptr -> user_data[0] =           DPN_SLV_NO_ACCESS;
dpn1_ptr -> user_data[1] =           DPN_SLV_NO_ACCESS;
dpn1_ptr -> user_data[2] =           DPN_SLV_NO_ACCESS;
dpn1_ptr -> user_data[3] =           DPN_SLV_READ;
dpn1_ptr -> user_data[4] =           DPN_SLV_WRITE_READ;
dpn1_ptr -> user_data[5] =           DPN_SLV_NO_ACCESS;
dpn1_ptr -> user_data[6] =           DPN_SLV_NO_ACCESS;
dpn1_ptr -> user_data[7] =           DPN_SLV_WRITE_READ;
error = dpn_init (dpn1_ptr);

dpn2_ptr -> reference.board_select = 2;
dpn2_ptr -> reference.access =       (DPN_SYS_CENTRAL) |
                                     (DPN_ROLE_CENTRAL);
dpn2_ptr -> length =                 4; // Index 0 to 3
dpn2_ptr -> user_data[0] =           DPN_SLV_NO_ACCESS;
dpn2_ptr -> user_data[1] =           DPN_SLV_NO_ACCESS;
dpn2_ptr -> user_data[2] =           DPN_SLV_NO_ACCESS;
dpn2_ptr -> user_data[3] =           DPN_SLV_WRITE_READ;
error = dpn_init (dpn2_ptr);
// Here, set the mode of Bus no. 2 using the
// function calls dpn_set_mode()
// and dpn_get_mode(). See Chapter 5 of the DP description
```

Example 3:
Single Board/
Multi-User Mode



One CP is installed in a computer. Three DP slaves are connected to the bus and have the L2 addresses 3, 4 and 7. DP slave no. 3 has only input ports, DP slave no. 4 has only output ports and DP slave no. 7 has both input and output ports.

Two DP Windows applications access the CP.

Following the first dpn_init() function call, the DP master will change automatically to the OPERATE mode.

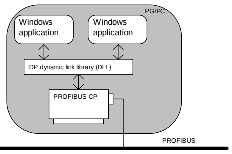
The dpn_ptr pointer points to the dpn_interface structure.

The initialization calls of the DP applications could appear as follows:

```
Application 1:

dpn_ptr -> reference.board_select =  1;
dpn_ptr -> reference.access =        (DPN_SYS_NOT_CENTRAL) |
                                     (DPN_ROLE_NOT_CENTRAL);
dpn_ptr -> length =                  8; // Index 0 to 7
dpn_ptr -> user_data[0] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[1] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[2] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[3] =            DPN_SLV_READ;
dpn_ptr -> user_data[4] =            DPN_SLV_WRITE_READ;
dpn_ptr -> user_data[5] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[6] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[7] =            DPN_SLV_READ;
error = dpn_init (dpn_ptr);

Application 2:

dpn_ptr -> reference.board_select =  1;
dpn_ptr -> reference.access =        (DPN_SYS_NOT_CENTRAL) |
                                     (DPN_ROLE_NOT_CENTRAL);
dpn_ptr -> length =                  8; // Index 0 to 7
dpn_ptr -> user_data[0] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[1] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[2] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[3] =            DPN_SLV_READ;
dpn_ptr -> user_data[4] =            DPN_SLV_READ;
dpn_ptr -> user_data[5] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[6] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[7] =            DPN_SLV_WRITE_READ;
error = dpn_init (dpn_ptr);
```

Remember that the right of access DPN_SLV_WRITE_READ for one DP slave can only be requested by one of the two DP applications.

Example 4: Single Board/Multi-User Mode



One CP is installed in a computer. Three DP slaves are connected to the bus and have the L2 addresses 3, 4 and 7. DP slave no. 3 has only input ports, DP slave no. 4 has only output ports and DP slave no. 7 has both input and output ports.

Two DP applications access the CP.

In contrast to example number 3, one of the two DP Windows applications will set the OPERATE mode itself. For this reason, the identifier (DPN_SYS_CENTRAL | DPN_ROLE_NOT_CENTRAL) is entered in the reference.access structure element.

The dpn_ptr pointer points to the dpn_interface structure.

The initialization calls of the DP Windows applications could appear as follows:

```
Application 1:

dpn_ptr -> reference.board_select =  1;
dpn_ptr -> reference.access =        (DPN_SYS_CENTRAL) |
                                     (DPN_ROLE_NOT_CENTRAL);
dpn_ptr -> length =                  8; // Index 0 to 7
dpn_ptr -> user_data[0] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[1] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[2] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[3] =            DPN_SLV_READ;
dpn_ptr -> user_data[4] =            DPN_SLV_WRITE_READ;
dpn_ptr -> user_data[5] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[6] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[7] =            DPN_SLV_READ;
error = dpn_init (dpn_ptr);

Application 2:

dpn_ptr -> reference.board_select =  1;
dpn_ptr -> reference.access =        (DPN_SYS_CENTRAL) |
                                     (DPN_ROLE_CENTRAL);
dpn_ptr -> length =                  8; // Index 0 to 7
dpn_ptr -> user_data[0] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[1] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[2] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[3] =            DPN_SLV_READ;
dpn_ptr -> user_data[4] =            DPN_SLV_READ;
dpn_ptr -> user_data[5] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[6] =            DPN_SLV_NO_ACCESS;
dpn_ptr -> user_data[7] =            DPN_SLV_WRITE_READ;
error = dpn_init (dpn_ptr);
// Here set the mode using the
// function calls dpn_set_mode()
// and dpn_get_mode().
```

9 Creating Unix Applications

Versions of Unix Supported

The following Unix variants are supported by the DP library:

- Interactive
- SCO
- Solaris

Remember that these variants have their own special features.

Compiler

The DP library was compiled under C++. To create UNIX applications, you therefore require either:

- a C++ compiler

  or

- the library **libC.a** of the C++ compiler and a standard C compiler.

Note:
There is no official C++ compiler for the Interactive variant. The freeware GNU compiler is, however available.

Libraries and Include Files Required

To create DP applications under Windows, you require the following files from the installation diskette.

Files of the Installation Diskette	File Name	Meaning
FDL Library	libfdl.a	FDL function prototypes
DP Library	libdp.a	DP function prototypes
DP Include File	dpn_user.h	DP-specific declarations and constants

Constants

When compiling the DP application, the define statement DPN_UNIX must be activated (for example with compiler switches).

☞ This define statement is required by the dpn_user.h include file and **must be provided by the user**.

Linker

When linking the DP application, make sure that the library is specified in the order libdp.a - libfdl.a.

Example: makefile

The following example shows a makefile for the **DP_Demo** program. It consists of a source file **DEMO.C** and the necessary libraries.

```
#-------------------------------------------------------------------
#  makefile : DP_Demo
#-------------------------------------------------------------------

#-------------------------------------------------------------------
#   Directives & Variables
#-------------------------------------------------------------------

.SILENT:
.SUFFIXES:     .C .o

CC             =  CC
LINKER         =  CC

CFLAGS         =  -DDPN_UNIX

OBJECTS =  demo.o
LIBS           =  libdp.a libfdl.a
DEMO           =  DP_Demo

#-------------------------------------------------------------------
#   Source Instructions
#-------------------------------------------------------------------

all:           $(DEMO)

$(DEMO):       makefile $(OBJECTS) $(LIBS)
               echo "creating $@ ..."
               $(CC) $(OBJECTS) $(LIBS) -o $(DEMO)
               echo "make was successful"

.C.o:
               echo "compiling $< ..."
               $(CC) $(CFLAGS) -c $<

#-------------------------------------------------------------------
#   End
#-------------------------------------------------------------------
```

Index

NOTES

Glossary

Base address
Logical address of a module in S7 systems.

Bus parameters
Bus parameters control the data transmission on the bus. Each -> station on the -> SINEC L2 network must use bus parameters that match those of other stations.

Bus segment
Part of a -> subnet. Subnets can consist of bus segments and connectivity devices such as repeaters and bridges.

CFB
Communication Function Block: A communication technique for program-controlled transmission of data from or to a CPU in an S7-300/400 using special function blocks. These function blocks were defined based on the IEC 1131-5 draft. The communication partners can be other modules with communication capabilities in an S7-300/400, operator stations, PCs or other controllers and computers.

COML DP
Configuration tool for configuring -> DP masters in -> SINEC L2.

CP
Communications Processor. Module for communication tasks.

Device master data
Device master data (DMD) contain DP slave descriptions complying with DIN E 19245 Part 3. Using DMD makes configuration of the -> DP master and -> DP slaves easier.

Distributed I/Os
Input and output modules used at a distance (distributed) from the CPU (central processing unit of the controller). The connection between the programmable controller and the distributed I/Os is established on -> SINEC L2. The programmable logic controllers do not recognize any difference between these I/Os and local process inputs and outputs.

DP I/O module
DP slaves have a modular design. A -> DP slave has at least one DP I/O module.

DP I/O type
The DP I/O type identifies a -> DP I/O module. The following types exist:
Input module
Output module
Input/output module

DP master
A -> station with master functions in -> SINEC L2 DP. The DP master controls the exchange of user data with the -> DP slaves assigned to it.

DP module list
The DP module list contains the modules belonging to a -> DP slave. You make entries in the DP module list when configuring a -> DP master with -> COML DP.

DP module name
Name of a -> DP I/O module entered in the ->DP module list.

DP module type	Type identifier of a -> DP I/O module in the -> device master data of a -> DP slave complying with DIN E 19245 Part 3.
DP slave	A -> station with slave functions in -> SINEC L2 DP.
DP slave catalog	The DP slave catalog contains the device descriptions of -> DP slaves required for configuring -> DP masters according to the -> DP standard. The DP slave catalog is available when configuring with -> COML DP.
DP slave name	A DP slave name is entered in the DP slave list to identify a -> DP slave in the DP configuration.
DP subnet	SINEC L2 subnet in which only -> distributed I/Os are operated.
DP subsystem	A -> DP master and all -> DP slaves with which this DP master exchanges data.
Driver	Software required for the data transfer between applications and the -> CP.
Enhanced mode	Enhanced mode under 3.x for personal computers with an Intel 386 or compatible processor.
FDL	Fieldbus Data Link. Layer 2 in -> PROFIBUS.
Frame	A message from one PROFIBUS station to another.
Frame header	A frame header consists of an identifier for the -> frame and the source and destination address.
Frame trailer	A frame trailer consists of a checksum and the end identifier of the -> frame.
FREEZE mode	The FREEZE mode is a DP mode in which process data are acquired at the same time and fetched from all (or a group of) DP slaves. The time at which the data are acquired is indicated in the FREEZE command (a synchronization control frame).
Gap update factor	A free address area (gap) between two active -> stations is checked cyclically by the station with the lower -> L2 address to find out whether or not another station is requesting to enter the logical ring. The cycle time for this check is as follows: gap update factor x target rotation time
Gateway	Intelligent connectivity device that connects different types of local area -> networks at OSI layer 7.
GD packet	Collection of data that may be distributed within the programmable logic controller (for example flags/memory bits or data blocks) to be transferred using the -> global data technique.

GD group
A GD group is a group of -> stations that exchange global data with each other. A -> GD packet is sent to the stations belonging to the GD group.

Global data
Global data (GD) is the name of a communication technique for the cyclic exchange of limited amounts of data from STEP 7 data areas between CPUs of the S7-300/400. Transmitted data can be received by several CPUs at the same time.

Global I/Os
Part of the I/O area of SIMATIC S5 PLCs can be used for global data exchange between SIMATIC S5 PLCs on -> SINEC L2. The main characteristic of this technique is the cyclic transmission of data that have changed since the last cycle.

Group identifier
DP slaves can be assigned to one or more groups using a group identifier. The -> control frames can be addressed to specific groups of DP slaves using the group identifier.

Highest L2 address
A -> bus parameter for -> SINEC L2. This specifies the highest -> L2 address (HSA) of an active -> station on the SINEC L2 bus. L2 addresses higher than the highest station address are possible for passive stations (possible values: HSA 1 to 126).

L2 address
The L2 address is a unique identifier for a -> station connected to -> SINEC L2 (PROFIBUS). The L2 address is transferred in the -> frame to address a -> station.

LSB
Least **S**ignificant **B**it.

Master
An active station in -> SINEC L2 that can send -> frames on its own initiative when it is in possession of the token.

Maximum station delay
A -> bus parameter for -> SINEC L2. The maximum station delay (max. TSDR) specifies the longest interval required by a -> station in the -> subnet between receiving the last bit of an unacknowledged -> frame and sending the first bit of the next frame. After sending an unacknowledged frame, a sender must wait for the max. TSDR to elapse before sending a further frame.

Minimum station delay
A -> bus parameter for -> SINEC L2. The minimum station delay (min. TSDR) specifies the minimum time that the receiver of a -> frame must wait before sending the acknowledgment or sending a new frame. The min. TSDR takes into account the longest interval required by a station in the subnet for receiving an acknowledgment after sending a frame.

MSB
Most **S**ignificant **B**it.

Network
A network consists of one or more interconnected -> subnets with any number of -> stations. Several networks can exist side by side. There is a common -> node table for every -> subnet.

Node table The node table applies to all -> networks within a -> system. Each entry in the node table describes the interface between a programmable logic controller (or any other station) and a -> subnet. The entries in the subnet are used by the system to locate and establish connections between stations.

Offset The length of the reserved area at the beginning of a data buffer of the FDL programming interface.

Process image The process image is a special memory area in the programmable logic controller. At the start of the cyclic program, the signal states of the input modules are transferred to the process image of the inputs. At the end of the cyclic program, the process image of the outputs is transferred to the output modules

PROFIBUS A fieldbus complying with DIN 19245.

PROFIBUS DP DP mode complying with DIN E 19245 Part 3.

PROFIBUS PA PROFIBUS PA is a recommendation of the PROFIBUS users' organization extending PROFIBUS DIN 19245 to include aspects of intrinsic safety.

Protocol A set of rules governing data transmission. Using these rules, both the formats of the messages and the data flow during transmission can be specified.

Reorganization token ring All the -> masters on -> SINEC L2 (PROFIBUS) form a logical token ring. Within this token ring, the token is passed on from station to station. If the transmission of the token is incorrect or if a master is removed from the ring, this leads to an error when the token is passed on (the token is not accepted by this station) and the station is excluded from the ring. The number of exclusions is counted in the internal token_error_counter. If this counter reaches an upper limit value, the logical token ring is then reorganized.

SCOPE L2 Diagnostic software for -> SINEC L2 with which the traffic on the -> network can be recorded and analyzed.

Segment Synonym for -> bus segment.

Services Services provided by a communication protocol.

Setup time A -> bus parameter for -> SINEC L2. The setup time specifies the minimum interval on the sender between receiving an acknowledgment and sending a new call frame.

SINEC Siemens Network and Communication. Product name for -> Siemens networks and network components.

SINEC L2 SINEC bus system for industrial applications based on PROFIBUS.

SINEC L2 DP SINEC L2 distributed I/Os. Transmission services complying with PROFIBUS DIN E 19245 Part 3.

SINEC L2 DP master	A -> station with master functions in -> SINEC L2 DP.
SINEC L2 FMS	SINEC L2 Fieldbus Message Specification. Upper sublayer of layer 7 of the ISO/OSI reference model for PROFIBUS.
Slot time	A bus parameter for -> SINEC L2. The slot time (TSL) is the time during which the sender of a -> frame waits for the acknowledgment from the receiver before detecting a timeout.
Station	A station is identified by an -> L2 address in the -> SINEC L2 network.
Subnet	A subnet is part of a -> network whose -> bus parameters (for example -> L2 addresses) must be matched. It includes the bus components and all attached stations. Subnets can, for example, be connected together by -> gateways to form a network. A -> system consists of several subnets with unique -> subnet numbers. A subnet consists of several ->stations with unique -> L2 addresses.
Subnet number	A -> system consists of several -> subnets with unique subnet numbers.
SYNC mode	The SYNC mode is a DP mode in which several or all -> DP slaves transfer data to their process outputs at a certain time. The time at which the data is transferred is indicated in the SYNC command (a control command for synchronization).
System	All the electrical equipment within a system. A system includes, among other things, programmable logic controllers, devices for operation and monitoring, bus systems, field devices, actuators, supply lines.
Target rotation time	A -> bus parameter for -> SINEC L2. The token represents the right to transmit for a -> station on SINEC L2. A station compares the actual token rotation time it has measured with the target rotation time and depending on the result can then send high or low priority frames.
Transmission rate	Transmission rate on the bus (unit in bits per second). A -> bus parameter for -> SINEC L2. The set or selected transmission rate depends on various conditions, for example distance across the network.
Watchdog	A monitoring time that can be set for a -> DP slave so that it detects the failure of the -> DP master to which it is assigned.

NOTES

❑