
We have checked the contents of this manual for agreement with the hardware described. Since deviations cannot be precluded entirely, we cannot guarantee full agreement. However, the data in this manual are reviewed regularly and any necessary corrections included in subsequent editions. Suggestions for improvement are welcome. Technical data subject to change.

Nous avons vérifié la conformité du contenu du présent manuel avec le matériel et le logiciel qui y sont décrits. Or, des divergences n'étant pas exclues, nous ne pouvons pas nous porter garants pour la conformité intégrale. Si l'usage du manuel devait révéler des erreurs, nous en tiendrons compte et apporterons les corrections nécessaires dès la prochaine édition. Veuillez nous faire part de vos suggestions. Nous nous réservons le droit de modifier les caractéristiques techniques.

Toute communication ou reproduction de ce support d'informations, toute exploitation ou communication de son contenu sont interdites, sauf autorisation expresse. Tout manquement à cette règle est illégal et expose son auteur au versement de dommages et intérêts. Tous nos droits sont réservés, notamment pour le cas de la délivrance d'un brevet ou celui de l'enregistrement d'un modèle d'utilité.

C79000-G8976-C126-05
Copyright © Siemens AG 1999 - 2005
All Rights Reserved
SIMATIC NET
DP Slave Programming Interface for CP 5511, CP 5512 and CP 5611 / CP 5611 A2

1 Introduction
2 Description of the Functions
3 Structure of a DP Slave Application
4 Additional DP Slave Information
5 Checklist for Programmers
6 Formats of the Slave Data
   Glossary
   Index
Classification of Safety-Related Notices

This document contains notices which you should observe to ensure your own personal safety, as well as to protect the product and connected equipment. These notices are highlighted in the manual by a warning triangle and are marked as follows according to the level of danger:

**Danger**
indicates that death or severe personal injury will result if proper precautions are not taken.

**Warning**
indicates that death or severe personal injury can result if proper precautions are not taken.

**Caution**
with warning triangle indicates that minor personal injury can result if proper precautions are not taken.

**Caution**
without warning triangle indicates that damage to property can result if proper precautions are not taken.

**Notice**
indicates that an undesirable result or status can occur if the relevant notice is ignored.

**Note**
highlights important information on the product, using the product, or part of the documentation that is of particular importance and that will be of benefit to the user.
Trademarks

SIMATIC®, SIMATIC NET® and SINEC® are registered trademarks of Siemens AG.

Third parties using for their own purposes any other names in this document which refer to trademarks might infringe upon the rights of the trademark owners.

Safety Instructions Regarding your Product

Before you use the product described here, read the safety instructions below thoroughly.

Qualified Personnel

Only qualified personnel should be allowed to install and work on this equipment. Qualified persons are defined as persons who are authorized to commission, to ground, and to tag circuits, equipment, and systems in accordance with established safety practices and standards.

Correct Usage of Hardware Products

Please note the following regarding the correct usage of hardware products:

Caution
This device and its components may only be used for the applications described in the catalog or the technical description, and only in connection with devices or components from other manufacturers which have been approved or recommended by Siemens.

This product can only function correctly and safely if it is transported, stored, set up, and installed correctly, and operated and maintained as recommended.

Before you use the supplied sample programs or programs you have written yourself, make certain that no injury to persons nor damage to equipment can result in your plant or process.

EU Directive: Do not start up until you have established that the machine on which you intend to run this component complies with the directive 89/392/EEC.
Correct Usage of Software Products

Please note the following regarding the correct usage of software products:

Caution
This software may only be used for the applications described in the catalog or the technical description, and only in connection with devices or software products from other manufacturers which have been approved or recommended by Siemens. Before you use the supplied sample programs or programs you have written yourself, make certain that no injury to persons nor damage to equipment can result in your plant or process.

Prior to Startup

Before putting the product into operation, note the following warning:

Caution
Prior to startup you must observe the instructions in the relevant documentation. For ordering data of the documentation, please refer to catalogs or contact your local Siemens representative.
# Contents

Note
The table of contents below only includes two levels to preserve clarity.
You will find a detailed table of contents at the beginning of each chapter.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>9</td>
</tr>
<tr>
<td>1.1</td>
<td>Classification</td>
<td>10</td>
</tr>
<tr>
<td>1.2</td>
<td>Properties of the Programming Interface</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>Description of the Functions</td>
<td>13</td>
</tr>
<tr>
<td>2.1</td>
<td>Introduction</td>
<td>14</td>
</tr>
<tr>
<td>2.2</td>
<td>Logging on a DP Slave Application – DPS_open</td>
<td>15</td>
</tr>
<tr>
<td>2.3</td>
<td>Logging off a DP Slave Application – DPS_close</td>
<td>19</td>
</tr>
<tr>
<td>2.4</td>
<td>Switching the Slave Online – DPS_start</td>
<td>20</td>
</tr>
<tr>
<td>2.5</td>
<td>Switching the Slave Offline – DPS_stop</td>
<td>21</td>
</tr>
<tr>
<td>2.6</td>
<td>Setting Input Data of the Slave (input data for master) – DPS_set_input</td>
<td>22</td>
</tr>
<tr>
<td>2.7</td>
<td>Getting the Output Data of the Slave (output data of the master) – DPS_get_output</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>Structure of a DP Slave Application</td>
<td>25</td>
</tr>
<tr>
<td>3.1</td>
<td>Requirements</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>Example of a DP Slave Application</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>Additional DP Slave Information</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>Evaluating Errors</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>Access to the CP 5511, Range of Functions of the DP Slave, GSD File</td>
<td>37</td>
</tr>
<tr>
<td>5</td>
<td>Checklist for Programmers</td>
<td>41</td>
</tr>
<tr>
<td>6</td>
<td>Formats of the Slave Data</td>
<td>43</td>
</tr>
<tr>
<td>6.1</td>
<td>Formats of the Input and Output Data</td>
<td>44</td>
</tr>
<tr>
<td>6.2</td>
<td>Formats of the Slave Diagnostic Data</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>Glossary</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>Index</td>
<td>53</td>
</tr>
</tbody>
</table>
Notes for the User

Documentation

This manual describes the DP slave programming interface.
1 Introduction

This description introduces the DP slave programming interface.
With this programming interface, a user program can use the services intended for a DP slave. The DP slave programming interface consists of functions belonging to the “C” programming language, grouped together to form a library (DLL).
The following section provides you with an overview of the properties and features of the DP slave programming interface.

---

Note
In this document, CP 5511 always means all the CPs from the SOFTNET family; in other words, CP 5511, CP 5512, CP 5611 and CP 5611 A2. Exception: If information relates to only one specific CP type, this is indicated explicitly.

Contents of the Chapter

1.1 Classification ........................................................................................................... 10
1.2 Properties of the Programming Interface ......................................................... 12
1.1 Classification

DP Master Class 1  A DP master class 1 communicates cyclically with the DP slaves. The communication includes central functions such as:

- Configuration and assignment of parameters to the slaves
- Cyclic data transfer (input/output) to the DP slaves
- Monitoring of the DP slaves
- Storage of diagnostic information

DP Master Class 2  With the CP 5511, access to the following diagnostic and parameter assignment functions are possible via a DP master class 2:

- Read slave diagnostic data
- Read master diagnostic data
- Read slave inputs
- Read save outputs
- Read slave configuration data
- Set slave address

DP Slave  The slave is activated via the DP slave programming interface, configured and assigned parameters by the DP master class 1 and it then exchanges input and output data with the DP master in cyclic data transfer (see DP master class 1).

- Set input data for DP master (max. 122 bytes)
- Read output data of the DP master (max. 122 bytes)
- Start/stop DP slave
Overview of the DP Protocol

The following diagram illustrates how a DP master class 2 is included in a PROFIBUS network.

Examples:
- Controller
- Pos. motor
- Sensor

DP Slave Programming Interface for CP 5511, CP 5512 and CP 5611 / CP 5611 A2

C79000-G8976-C126-05
# 1.2 Properties of the Programming Interface

<table>
<thead>
<tr>
<th>Property</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Names</strong></td>
<td>The names of the functions and data components are based on the DP specification.</td>
</tr>
<tr>
<td><strong>Single User Operation</strong></td>
<td>The programming interface is designed for a call within one thread within a user program. Multi-user operation is not possible.</td>
</tr>
<tr>
<td><strong>Multiboard Operation</strong></td>
<td>Not approved.</td>
</tr>
<tr>
<td><strong>Operating Systems and Compilers</strong></td>
<td>The programming interface is designed for use with Windows 98, Windows NT 4.0, Windows 2000 Professional, Windows XP and Windows Server 2003. To find out which compiler or development environment is currently supported under which operating system, refer to the section &quot;Supported Development Environments or Compilers&quot; in the &quot;Readme.htm&quot; file of the relevant SIMATIC NET CD.</td>
</tr>
<tr>
<td><strong>Synchronous and Parallel Processing</strong></td>
<td>The function calls of the programming interface are synchronous. Synchronous functions only return to the caller after the job has been executed and provide the results of the job. The parallel execution of more than one DP slave service is not possible and is rejected with an error.</td>
</tr>
<tr>
<td><strong>Supported Data Type</strong></td>
<td>The byte data type is supported.</td>
</tr>
<tr>
<td><strong>Consistency</strong></td>
<td>Byte consistency is supported.</td>
</tr>
<tr>
<td><strong>Maximum Data Lengths</strong></td>
<td>A maximum of 122 bytes are possible in both the input and output direction.</td>
</tr>
<tr>
<td><strong>Compatibility with the CP 5614</strong></td>
<td>The call interface is compatible with the CP 5614, however, it provides only a subset of the CP 5614 slave functions.</td>
</tr>
</tbody>
</table>
| **Supported Boards**                          | - CP 5511  
  - CP 5512  
  - CP 5611  
  - CP 5611 A2 |
2 Description of the Functions

This chapter describes the administrative and productive functions of the DP slave programming interface.

Contents of the Chapter

2.1 Introduction ........................................................................................................14
2.2 Logging on a DP Slave Application – DPS_open..............................................15
2.3 Logging off a DP Slave Application – DPS_close .............................................19
2.4 Switching the Slave Online – DPS_start ...........................................................20
2.5 Switching the Slave Offline – DPS_stop ...........................................................21
2.6 Setting Input Data of the Slave (input data for master) – DPS_set_input........22
2.7 Getting the Output Data of the Slave (output data of the master) – DPS_get_output...........................................................................................................................23
2.1 Introduction

The DP slave programming interface consists of a number of C functions that can be grouped as follows:

- Administrative Functions
- Productive functions (functions for data exchange)

### Administrative Functions

The following table illustrates the calls for administrative functions.

<table>
<thead>
<tr>
<th>Administrative Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPS_open</td>
<td>DP slave user program logs on at the interface.</td>
</tr>
<tr>
<td>DPS_close</td>
<td>User program logs off at the interface.</td>
</tr>
<tr>
<td>DPS_stop</td>
<td>Deactivates the slave module (offline).</td>
</tr>
<tr>
<td>DPS_start</td>
<td>Activates the slave module (online).</td>
</tr>
</tbody>
</table>

### Functions for Data Exchange

The following table illustrates the additional functions for data exchange.

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPS_set_input</td>
<td>Set input data of the slave (input data for the DP master)</td>
</tr>
<tr>
<td>DPS_get_output</td>
<td>Read the output data of the slave. (output data from DP master)</td>
</tr>
</tbody>
</table>

### Conventions used in Notation

The following table explains the meaning of the special representations in the following sample programs for the functions:

<table>
<thead>
<tr>
<th>Representation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>// in</td>
<td>The value is provided by the user program as input to the DP slave programming interface.</td>
</tr>
<tr>
<td>// out</td>
<td>The value is returned to the user program by the DP slave programming interface.</td>
</tr>
</tbody>
</table>
2.2  Logging on a DP Slave Application – DPS_open

Purpose
With this function, the a DPS application logs on at the driver and sets
the slave parameters. If successful, the function returns a user handle.
The user handle must be included in all further function calls.
The bit "DPS_SM_SIMPLE" is set in the parameter "slave_mode"; the
expected parameter and configuration data are located in the para-
meter "init_data".

Syntax

```
DPR_DWORD DPS_open (  
DPR_STRING *cp_name,   // in  
DPR_DWORD *user_handle, // out  
DPR_DWORD slave_mode,   // in  
DPRWORD station_addr,   // in  
DPRWORD addr_change,    // in, not used!  
DPRWORD pno_ident_nr,   // in  
DPRWORD user_wd,        // in, not used!  
DPS_INIT_DATA_T *init_data // in  
DPS_MAX_DATA_T *max_data_len // in, not used!  
DPRWORD baud_rate        // in  
DP_ERROR_T *error);      // out
```

Example

```
if (  DPS_open("DPSONLINE",&user_handle,  
DPS_SM_SIMPLE, (DPR_BYTE)StationAddr, 0, 0x8076,  
0,&init_data,NULL, baudrate, &error) == DP_OK)  
{ // OK. Further actions }
else
{ // Error handling }
```
### Description of the Functions

#### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cp_name</td>
<td>Access point to the hardware - Using the program &quot;Set PG/PC Interface&quot;, a &quot;CP5511 PROFIBUS DP SLAVE&quot; interface parameter set must be assigned to this access point.</td>
</tr>
<tr>
<td>user_handle</td>
<td>Pointer to the user-handle variable - If successful, the user handle assigned to the application is entered here.</td>
</tr>
<tr>
<td>slave_mode</td>
<td>DPS_SM_SIMPLE</td>
</tr>
<tr>
<td>station_addr</td>
<td>Station address of the slave</td>
</tr>
<tr>
<td>addr_change</td>
<td>Always 0: address changes via the bus are not permitted</td>
</tr>
<tr>
<td>pno_ident_nr</td>
<td>Unique number for the slave assigned by the PROFIBUS Users Organization (for example 0x0008 for the sample application) when the customer applies for certification.</td>
</tr>
<tr>
<td>user_wd</td>
<td>No significance</td>
</tr>
</tbody>
</table>

Table continued on next page
Table continued from previous page

<table>
<thead>
<tr>
<th>init_data</th>
<th>Pointer to a structure with information on the extended slave data.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Typedef union DPS_INIT_DATA_S</strong></td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>struct DPS_SIMPLE_S</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>DPR_WORD user_prm_data_len;</td>
<td>// in</td>
</tr>
<tr>
<td>DPR_BYTE user_prm_data[DPS_MAX_PDU_LEN];</td>
<td>// in</td>
</tr>
<tr>
<td>DPR_WORD cfg_data_len;</td>
<td>// in</td>
</tr>
<tr>
<td>DPR_BYTE cfg_data[DPS_MAX_PDU_LEN];</td>
<td>// in</td>
</tr>
<tr>
<td>}simple;</td>
<td></td>
</tr>
<tr>
<td>struct DPS_DYNAMIC_S</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>DPR_WORD def_cfg_data_len;</td>
<td>// in</td>
</tr>
<tr>
<td>DPR_BYTE def_cfg_data[DPS_MAX_PDU_LEN];</td>
<td>// in</td>
</tr>
<tr>
<td>}dynamic;</td>
<td></td>
</tr>
<tr>
<td>)DPS_INIT_DATA_T</td>
<td></td>
</tr>
<tr>
<td><strong>user_prm_data_len:</strong></td>
<td>Length of the specified user parameter assignment data : always 0 !</td>
</tr>
<tr>
<td><strong>user_prm_data:</strong></td>
<td>Specified user parameter assignment data : always 0 !</td>
</tr>
<tr>
<td><strong>cfg_data_len:</strong></td>
<td>Length of the specified configuration data</td>
</tr>
<tr>
<td><strong>cfg_data:</strong></td>
<td>Predefined configuration data from the GSD file</td>
</tr>
<tr>
<td><strong>def_cfg_data_len:</strong></td>
<td>No significance, (struct DPS_DYNAMIC_S not used)</td>
</tr>
<tr>
<td><strong>def_cfg_data:</strong></td>
<td>No significance, (struct DPS_DYNAMIC_S not used)</td>
</tr>
</tbody>
</table>

* user_prm_data are data specific to the application and are appended to the standard section of parameter data (for example to set a mode, measuring range etc. on the slave). With a SIMPLE_SLAVE, the user parameter data of the master are simply compared with the specified parameter data.
Table continued from previous page

<table>
<thead>
<tr>
<th>max_data_len</th>
<th>The structure is not evaluated by the CP 5511 DP slave and is therefore of no significance.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>typedef DPS_MAX_DATA_S</code></td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>DPR_BYTE max_input_data_len; // in</td>
</tr>
<tr>
<td></td>
<td>DPR_BYTE max_output_data_len; // in</td>
</tr>
<tr>
<td></td>
<td>DPR_BYTE max_user_diag_len; // in</td>
</tr>
<tr>
<td></td>
<td>DPR_BYTE max_user_prm_data_len; // in</td>
</tr>
<tr>
<td></td>
<td>DPR_BYTE max_cfg_data_len; // in</td>
</tr>
<tr>
<td></td>
<td>DPR_BYTE max_user_ssa_data_len; // in</td>
</tr>
<tr>
<td></td>
<td>}DPS_MAX_DATA_T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>baud_rate</th>
<th>Transmission rates that can be set: The following constants are defined:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>DPS_BD_9K6</code> // 9.6 Kbps</td>
</tr>
<tr>
<td></td>
<td><code>DPS_BD_19K2</code> // 19.2 Kbps</td>
</tr>
<tr>
<td></td>
<td><code>DPS_BD_45K45</code> // 45.45 Kbps</td>
</tr>
<tr>
<td></td>
<td><code>DPS_BD_93K75</code> // 93.75 Kbps</td>
</tr>
<tr>
<td></td>
<td><code>DPS_BD_187K5</code> // 187.5 Kbps</td>
</tr>
<tr>
<td></td>
<td><code>DPS_BD_500K</code> // 500 Kbps</td>
</tr>
<tr>
<td></td>
<td><code>DPS_BD_1M5</code> // 1.5 Mbps</td>
</tr>
<tr>
<td></td>
<td><code>DPS_BD_3M</code> // 3 Mbps</td>
</tr>
<tr>
<td></td>
<td><code>DPS_BD_6M</code> // 6 Mbps</td>
</tr>
<tr>
<td></td>
<td><code>DPS_BD_12M</code> // 12 Mbps</td>
</tr>
</tbody>
</table>

| error        | Address of a structure provided by the user program of the type DP_ERROR_T. If an error occurred, the structure contains details for troubleshooting. |

Return Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP_OK</td>
<td>Successful completion of the function</td>
</tr>
<tr>
<td>other</td>
<td>Incorrect completion - the error IDs correspond to the function. For more detailed information, refer to Chapter 4.</td>
</tr>
</tbody>
</table>

Note
Make sure that the node address of the slave does not already exist on the bus.
This function is synchronous.
The PROFIBUS address of the slave and the baud rate of the bus transferred with the DPS_open function are ignored. The existing settings for these parameters are retained.
2.3 Logging off a DP Slave Application – DPS_close

**Purpose**

With this function, a DP user program logs off for communication at the slave module.

**Syntax**

```c
DPR_DWORD DPS_close(
    DPR_DWORD   user_handle,  // in
    DP_ERROR_T  *error );     // out
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_handle</td>
<td>User handle assigned with the DPS_open call.</td>
</tr>
<tr>
<td>error</td>
<td>Address of a structure provided by the user program of the type DP_ERROR_T. If an error occurred, the structure contains details for troubleshooting.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP_OK</td>
<td>Successful completion of the function</td>
</tr>
<tr>
<td>other</td>
<td>Incorrect completion - the error IDs correspond to the function. For more detailed information, refer to Chapter 4.</td>
</tr>
</tbody>
</table>

**Note**

- If the logoff was successful, the user handle is no longer valid and must not continue to be used.
- This function is synchronous.
2.4 Switching the Slave Online – DPS_start

**Purpose**

- This function can be used to switch the slave online. This is necessary after initialization. After switching the slave OFFLINE with DPS_stop, the slave module can be switched online again with DPS_start.

- Following the DPS_start call, the DP slave is ready to receive the parameter assignment and configuration frames of the DP masters and to change to the DATA_EXCHANGE state after successful parameter assignment and configuration. If DPS_get_output or DPS_set_input calls are sent immediately following DPS_start, but the DP slave is still in the startup phase, these calls will be acknowledged with DPS_ERROR_EV_NO_DATA_EX (see Section 4.1.1) until the startup phase is completed and the slave changes to the DATA_EXCHANGE state.

**Syntax**

```c
DPR_DWORD DPS_start (  
    DPR_DWORD user_handle, // in  
    DP_ERROR_T *error);  // out
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_handle</td>
<td>User handle assigned with the DPS_open call.</td>
</tr>
<tr>
<td>error</td>
<td>Address of a structure provided by the user program of the type DP_ERROR_T. If an error occurred, the structure contains details for troubleshooting.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP_OK</td>
<td>Successful completion of the function</td>
</tr>
<tr>
<td>other</td>
<td>Incorrect completion - the error IDs correspond to the function. For more detailed information, refer to Chapter 4.</td>
</tr>
</tbody>
</table>

**Note**

This function is synchronous.
2.5 Switching the Slave Offline – DPS_stop

**Purpose**

This function is used to deactivate the slave module. The slave module then no longer reacts on the bus. To reactivate the slave, the DPS_start function must be used.

**Note**

The DPS_stop function starts to shut down the DP slave. It is possible that the function is acknowledged with DP_OK, although the slave has not yet completed the change to the OFFLINE state.

**Syntax**

```c
DPR_DWORD DPS_stop (  
    DPR_DWORD user_handle, // in  
    DP_ERROR_T *error ); // out
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_handle</td>
<td>User handle assigned with the DPS_open call.</td>
</tr>
<tr>
<td>error</td>
<td>Address of a structure provided by the user program of the type DP_ERROR_T. If an error occurred, the structure contains details for troubleshooting.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP_OK</td>
<td>Successful completion of the function</td>
</tr>
<tr>
<td>other</td>
<td>Incorrect completion - the error IDs correspond to the function. For more detailed information, refer to Chapter 4.</td>
</tr>
</tbody>
</table>

**Note**

This function is synchronous.
2.6 Setting Input Data of the Slave (input data for master) – DPS_set_input

Purpose
This function sets input data of a slave (input data for the DP master). If this service is sent immediately after a DPS_start, it is possible that the slave has not yet reached the DATA_EXCHANGE state, in other words, the parameter assignment and configuration by the DP master is not yet completed and data exchange is not yet possible. The call is acknowledged during this time with DPS_ERROR_EV_NO_DATA_EX (see also Section 4.1.1).

Data Consistency
Byte consistency is guaranteed for data input.

Syntax

```
DPR_DWORD DPS_set_input(
    DPR_DWORD user_handle, // in
    DPR_BYTE *data_s, // in
    DPR_WORD len, // in
    DP_ERROR_T *error ); // out
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_handle</td>
<td>User handle assigned with the DPS_open call.</td>
</tr>
<tr>
<td>data_s</td>
<td>Input data of the slave</td>
</tr>
<tr>
<td>len</td>
<td>Data length (maximum 122 bytes)</td>
</tr>
<tr>
<td>error</td>
<td>Address of a structure provided by the user program of the type DP_ERROR_T. If an error occurred, the structure contains details for troubleshooting.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP_OK</td>
<td>Successful completion of the function</td>
</tr>
<tr>
<td>other</td>
<td>Incorrect completion - the error IDs correspond to the function. For more detailed information, refer to Chapter 4.</td>
</tr>
</tbody>
</table>

Note
This function is synchronous.
Description of the Functions

2.7 Getting the Output Data of the Slave (output data of the master) – DPS_get_output

Purpose
This function gets the output data of the slave (output data of the DP master).

If this service is sent immediately after a DPS_start, it is possible that the slave has not yet reached the DATA_EXCHANGE state, in other words, the parameter assignment and configuration by the DP master is not yet completed and data exchange is not yet possible. The call is acknowledged during this time with DPS_ERROR_EV_NO_DATA_EX (see also Section 4.1.1).

Data Consistency
Byte consistency is guaranteed for data output.

Syntax

```c
DPR_DWORD  DPS_get_output
   (DPR_DWORD   user_handle,   // in
    DPR_BYTE   *data_s,       // out
    DPR_WORD   len,           // in
    DP_ERROR_T *error );      // out
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_handle</td>
<td>User handle assigned with the DPS_open call.</td>
</tr>
<tr>
<td>data_s</td>
<td>Output data of the slave</td>
</tr>
<tr>
<td>len</td>
<td>Data length (maximum 122 bytes)</td>
</tr>
<tr>
<td>error</td>
<td>Address of a structure provided by the user program of the type DP_ERROR_T. If an error occurred, the structure contains details for troubleshooting.</td>
</tr>
</tbody>
</table>
Description of the Functions

Return Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP_OK</td>
<td>Successful completion of the function</td>
</tr>
<tr>
<td>other</td>
<td>Incorrect completion - the error IDs correspond to the function. For more detailed information, refer to Chapter 4.</td>
</tr>
</tbody>
</table>

**Note**

This function is synchronous.
3 Structure of a DP Slave Application

This chapter describes the following:
- Why a GSD file is required
- How a DP slave application can be structured
- Call sequences of DP slave services

Contents of the Chapter

3.1 Requirements .................................................................................................... 26
3.2 Example of a DP Slave Application ................................................................... 27
3.1 Requirements

GSD File

A GSD file must be provided by the developer of a DP slave to describe the properties of the product.

It is necessary both for configuration and certification. The file format specified for the GSD files (GSD is an acronym from the German meaning default device database) allows open configuration tools to be implemented for PROFIBUS DP.

The specified keywords and the fixed structure of a GSD file can be found in the PROFIBUS standard EN 50170, Part 2, or in the appropriate literature.

Caution

Please note the following:

The GSD file supplied with the demonstration program contains the description of the demonstration slave (with fixed configuration parameters). The ID number is a default number reserved for SIEMENS. Its use is permitted only for test purposes with the demonstration application.

For more detailed information on the structure of a GSD file for the DP slave in SOFTNET, refer to Section 4.2

GSD File Preparation

To implement "your" slave with this programming interface, you must describe the slave with its GSD file.

GSD File and Configuration Tool

The GSD file must be made known to your configuration tool for example SIEMENS COM PROFIBUS).

For more information, refer to the documentation of your configuration tool.

Then create a database containing the slave and configure your PROFIBUS DP network with this database.

Certification

If you want to certify your DP slave (with the corresponding GSD file), you must first apply for a free identification number from the PNO.
3.2 Example of a DP Slave Application

The following section outlines the basic structure of a DP slave application.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bind include files and libs</td>
</tr>
<tr>
<td>2</td>
<td>Create variables</td>
</tr>
<tr>
<td>3</td>
<td>Log on with the DP slave LIB (DPS_open)</td>
</tr>
<tr>
<td>4</td>
<td>Start the slave (switch ONLINE) (DPS_start)</td>
</tr>
<tr>
<td>5</td>
<td>Send one or more DP slave jobs</td>
</tr>
<tr>
<td>6</td>
<td>Stop slave (switch OFFLINE) (DPS_stop)</td>
</tr>
<tr>
<td>7</td>
<td>Log off from the DP slave LIB (DPS_close)</td>
</tr>
</tbody>
</table>

**Step 1** Bind include files and libs
Include the header files.
You must also make sure that the dpslib.lib is linked to your project.

```c
#include "dps_fkt.h"  // includes dps_user.h too
```

**Step 2** Create variables
Creating data buffers and structures:

```c
DPR_DWORD        user_handle;
DPR_WORD         station_address;
DPR_WORD         pno_ident_number = 1234;
                 //example.
DP_ERROR_T       error;
DPS_INIT_DATA_T  init_data;
DPR_WORD         baudrate = 0x03;
```
Step 3

Log on at the DP slave LIB

Before you can use the DP slave services, you must log on at the DP slave lib.

```c
error.error_code = 0;
// example configuration
init_data.simple.cfg_data[0] = 0x3f;
init_data.simple.cfg_data[1] = 0x35;
init_data.simple.cfg_data[2] = 0x1f;
init_data.simple.cfg_data[3] = 0x23;
// The GSD file must include:
// Module = "module1" 0x3f, 0x35, 0x1f, 0x23
// see Section 4.2
init_data.simple.cfg_data_len = 4;
init_data.simple.user_prm_data_len = 0; // always 0!
init_data.simple.user_prm_data = 0;     // always 0!
// Example here:
// 0x3f = 16 inputs / 16 outputs
// 0x35 =  6 inputs /  6 outputs
// 0x1f = 16 inputs /   0 outputs
// 0x23 =  0 inputs /   4 outputs
// Total 38 inputs / 26 outputs
// Bit coding of .cfg_data:
// Bit 0 to 3 = length+1
// Bit 4 = 1: input
// Bit 5 = 1: output
// Bit 6, 7 always 0
.
.
.
if ( DPS_open ("DPSONLINE", &user_handle,
DPS_SM_SIMPLE, (DPR_BYTE)StationAddr, 0,
no_ident_number, 0, &init_data, NULL, baudrate,
&error) == DP_OK)
{ // OK. Further actions }
else
{ // Error handling }
```
Structure of a DP Slave Application

Note
The values in init_data.simple.cfg_data[n] must match the values in the GSD file.
The lengths specified for the services DPS_get_output and DPS_set_input must match the values of the configuration parameters and GSD file exactly. In other words, in the example above DPS_get_output(..., len,...) with len = 26 and DPS_set_input(...,len,...) with len = 38.

Step 4
Start the slave (switching online)
Send DPS_start.

```c
DP_ERROR_T error;
error.error_code = 0;
if ( DPS_start(user_handle, &error) == DP_OK )
    {........} // OK. Further actions
else {........} // Error handling
```

Step 5
Send a DP slave job, for example read output data

```c
DP_ERROR_T error;
DPR_BYTE   OutputData[OUTPUT_SIZE];
DPR_WORD   len;

error.error_code = 0;
if (DPS_get_output(user_handle, OutputData, len, &error) == DP_OK)
    {........} // OK. Further actions
else {........} // Error handling
```

Step 6
Stop slave (switch offline)

```c
DP_ERROR_T error;
error.error_code = 0;
if ( DPS_stop(user_handle, &error) == DP_OK )
    {........} // OK. Further actions
else {........} // Error handling
```
Step 7  

Log off from the DP slave LIB

Before you can terminate the application, it must log-off:

```c
DP_ERROR_T error;
error.error_code = 0;
if ( DPS_close(user_handle, &error) == DP_OK )
    {.......} // OK. Further actions
else {.......} // Error handling
```

Caution

**Please note the following:** Check the return values of the DP slave functions for any errors.
4 Additional DP Slave Information

This chapter describes the following:

- The meaning of the error bits and how to evaluate them
- General information on the range of functions of the DP slave
- General information on the operating systems
- General information on the administrative services

Contents of the Chapter

4.1 Evaluating Errors ............................................................................................... 32
4.1.1 Entries in the error_code and error_decode Structure Elements.................. 34
4.2 Access to the CP 5511, Range of Functions of the DP Slave, GSD File......... 37
4.1 Evaluating Errors

Purpose
The error identifiers for the individual jobs have a uniform structure DP_ERROR. If an error occurs, the various elements return the exact description of the error. The declarations are in the include file "dps_user.h".

Syntax
```c
typedef struct DP_ERROR
{
    DPR_DWORD error_class;
    DPR_DWORD error_code;
    DPR_BYTE error_decode;
    DPR_BYTE error_code_1;
    DPR_BYTE error_code_2;
} DP_ERROR_T;
```

error_class Structure Element
The error_class structure element specifies the general error class. The entry in error_class is identical to the return value of the function call. A DP slave application can therefore use either the error_class structure element or the return value of the function call.

The possible error classes are listed in Table 4-1.

error_code Structure Element
The error_code structure element is relevant in the error classes:
- DP_ERROR_EVENT_NET
- DP_ERROR_REQ_PAR
- DP_ERROR_RES

error_decode Structure Element
The error_decode structure element is relevant in the error classes:
- DP_ERROR_RES
- DP_ERROR_REQ_PAR

Structure Elements
error_code_1, error_code_2
These elements are currently not used.
How to Evaluate Errors

The following table shows which entries are returned by a job in the error_class structure element.

In some situations, further information can be evaluated

- In the error_code structure elements.
- In the error_decode structure element.

Description of Table 4-1

The following Table 4-1 describes the entries in the error_class structure element.

<table>
<thead>
<tr>
<th>Entry in the error_class Structure Element</th>
<th>Description</th>
<th>Further Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP_ERROR_EVENT_NET</td>
<td>No communication possible with the slave</td>
<td>Table 4-2</td>
</tr>
<tr>
<td>DP_ERROR_REQ_PAR</td>
<td>Incorrect transfer parameter of a call or call illegal</td>
<td>Table 4-3</td>
</tr>
<tr>
<td>DP_ERROR_RES</td>
<td>Not enough resources from system or layer 2</td>
<td>Table 4-4</td>
</tr>
<tr>
<td>DP_OK</td>
<td>Job completed without error</td>
<td></td>
</tr>
</tbody>
</table>

Table 4-1 Explanation of the error_class Structure Element
4.1.1 Entries in the error_code and error_decode Structure Elements

The following Table 4-2 describes the entries in the error_code structure element. These belong to the error codes of the DP_ERROR_EVENT_NET class.

<table>
<thead>
<tr>
<th>Entry in error_code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPS_ERROR_EV_NO_DATA_EX</td>
<td>DP slave is not in the DATA_EXCHANGE state</td>
</tr>
<tr>
<td>DPS_ERROR_EV_START_FAILED</td>
<td>Start command could not be executed</td>
</tr>
<tr>
<td>DPS_ERROR_SIM9SYNC_FAILED</td>
<td>There may be a problem in your installation. Repeat the installation and if</td>
</tr>
<tr>
<td></td>
<td>the problem recurs, contact our hotline.</td>
</tr>
<tr>
<td>DPS_ERROR_EV_COMM_ERR</td>
<td>Bad SCP_Send. For example, DPS_open with bus short-circuit (physical bus</td>
</tr>
<tr>
<td></td>
<td>defect).</td>
</tr>
</tbody>
</table>

Table 4-2 Error Coding of the DP_ERROR_EVENT_NET Class
Description of Table 4-3

The following Table 4-3 describes the entries in the error_code structure element. These belong to the error codes of the DP_ERROR_REQ_PAR class.

<table>
<thead>
<tr>
<th>Entry in error_code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPS_ERROR_PAR_INTERNAL *</td>
<td>Internal error. Please call the Hotline.</td>
</tr>
<tr>
<td>DPS_ERROR_PAR_SLAVE_MODE</td>
<td>Slave mode not permitted</td>
</tr>
<tr>
<td>DPS_ERROR_PAR_ADDR_CHANGE</td>
<td>ADDR_CHANGE not permitted</td>
</tr>
<tr>
<td>DPS_ERROR_PAR_USER_HANDLE</td>
<td>Invalid user handle</td>
</tr>
<tr>
<td>DPS_ERROR_PAR_STATION_ADDR</td>
<td>Station address higher than 126 not permitted</td>
</tr>
<tr>
<td>DPS_ERROR_PAR_INPUT_DATA</td>
<td>No pointer to input data</td>
</tr>
<tr>
<td>DPS_ERROR_PAR_OUTPUT_DATA</td>
<td>No pointer to output data</td>
</tr>
<tr>
<td>DPS_ERROR_PAR_BAUDRATE</td>
<td>Wrong transmission rate</td>
</tr>
<tr>
<td>DPS_ERROR_REQ_NOT_ALLOWED</td>
<td>Request not permitted in this state</td>
</tr>
<tr>
<td>DPS_ERROR_REQ_DPS_ACTIVE</td>
<td>Request not permitted temporarily since active request not completed.</td>
</tr>
<tr>
<td>DPS_ERROR_PAR_OUTPUT_LEN</td>
<td>DPS_get_output (): &quot;Output data length&quot; parameter does not match the length specified in the configuration data.</td>
</tr>
<tr>
<td>DPS_ERROR_PAR_INPUT_LEN</td>
<td>DPS_set_input (): &quot;Input data length&quot; parameter does not match the length specified in the configuration data.</td>
</tr>
<tr>
<td>DPS_ERROR_PAR_IO_LEN</td>
<td>DPS_open (): Too many I/Os; maximum 122 permitted.</td>
</tr>
<tr>
<td>DPS_ERROR_PAR_CFG_DATA **</td>
<td>Bad configuration data:</td>
</tr>
<tr>
<td></td>
<td>No outputs/inputs configured</td>
</tr>
</tbody>
</table>

* Additional entry in error_decode for DPS_ERROR_PAR_INTERNAL

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPS_ERROR_INTERNAL_SCP_PARAM</td>
</tr>
<tr>
<td>DPS_ERROR_INTERNAL_AMPRO2_LOAD</td>
</tr>
<tr>
<td>DPS_ERROR_INTERNAL_AMPRO2_OPEN</td>
</tr>
</tbody>
</table>

Table continued on next page
** Additional entry in error_decode for DPS_ERROR_PAR_CFG_DATA

<table>
<thead>
<tr>
<th>Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>error_decode contains the index of the bad CFG byte (bit 4 or 5 in the CFG byte not set)</td>
</tr>
</tbody>
</table>

For more detailed information, refer to the frequently asked questions (FAQ) for your product or contact your Siemens representative.

** Table 4-3 Error Coding of the DP_ERROR_REQ_PAR Class

** Description of Table 4-4  **

The following Table 4-4 describes the entries in the error_code structure element. These belong to the error codes of the DP_ERROR_RES class.

<table>
<thead>
<tr>
<th>Entry in error_code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPS_ERROR_RES_OPEN</td>
<td>No further SCP_OPEN possible</td>
</tr>
<tr>
<td>DPS_ERROR_RES_ALLOC *</td>
<td>Error in layer 2 resource or system resource allocation</td>
</tr>
<tr>
<td>DPS_ERROR_RES_DEALLOC **</td>
<td>Error in layer 2 resource or system resource deallocation</td>
</tr>
<tr>
<td>DPS_ERROR_RES_REG_BAUDRATE</td>
<td>Transmission rate could not be written to the registry.</td>
</tr>
<tr>
<td>DPS_ERROR_RES_REG_TS</td>
<td>Station address could not be written to the registry.</td>
</tr>
<tr>
<td>DPS_ERROR_RES_REG_LOG_DEVICE</td>
<td>Logical device or name not found in the registry.</td>
</tr>
<tr>
<td>DPS_ERROR_RES_REG_T_SLOT</td>
<td>Writing the bus parameter Tslot to the registry failed</td>
</tr>
</tbody>
</table>

* Additional entry in error_decode for DPS_ERROR_RES_ALLOC

<table>
<thead>
<tr>
<th>Entry in error_code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPS_ERROR_RES_ALLOC_SYS</td>
<td>Error in system resource allocation</td>
</tr>
<tr>
<td>DPS_ERROR_RES_ALLOC_AMPRO2</td>
<td>Error in layer 2 resource allocation</td>
</tr>
</tbody>
</table>

** Additional entry in error_decode for DPS_ERROR_RES_DEALLOC

<table>
<thead>
<tr>
<th>Entry in error_code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPS_ERROR_RES_DEALLOC_SYS</td>
<td>Error in system resource deallocation</td>
</tr>
<tr>
<td>DPS_ERROR_RES_DEALLOC_AMPRO2</td>
<td>Error in layer 2 resource deallocation</td>
</tr>
</tbody>
</table>
4.2 Access to the CP 5511, Range of Functions of the DP Slave, GSD File

Board
With the DPS_open function, an access point is selected in the "cp_name" parameter. For example, "DPSONLINE" or any other string (you can also define your own string).

Settings in Set PG/PC Interface
A CP 5511 interface parameter set must be assigned to the access point in the "Set PG/PC Interface" tool.

Example
"CP 5511 (PROFIBUS DP Slave)".
This assigns a CP 5511 PCMCIA card to your DP slave.

Further Settings
No further settings are necessary with the "Set PG/PC Interface" tool.
All the remaining parameters necessary for operating the slave are transferred using the "DPS_open (params)" service.

Range of Functions
The CP 5511 DP slave provides you with a simple and clear programming interface, with which you can create an application with little effort.
Your slave application can be certified in the PROFIBUS interface center (SSC) in Fürth, Germany.
The interface of the CP 5511 DP slave does not currently provide the following (optional) functions:
1. DP-V1 functions
2. automatic data transmission rate recognition
3. SYNC/FREEZE

GSD File
A GSD file describes the functional properties of a DP slave. The specified keywords and the fixed structure of a GSD file can be found in the PROFIBUS standard EN 50170 (Volume 2), or in the appropriate literature.
If you create your own GSD file, base it on the sample file supplied.
Creating and modifying GSD files

When creating a GSD file for a SOFTNET DP slave, note the following points:

- In the configuration data of the submodule definition, only use "bytes" for the length of the data units.
- In the configuration data of the submodule definition, only use "byte consistency" for the consistency.
- Do not enter any empty submodules.
- The maximum data length per submodule is 16 bytes.
- The total length of all inputs or outputs is 122 bytes.
- If you use the GSD file "siem8076.gsd", remember that this is only an example. If you want to create a user program that uses the OPC interface, you must change the Ident number in the GSD file from Ident_Number=0x8076 to 0x9001.
- Fixed values must not be changed.
- Changes can lead to system malfunctions!
- Do not modify any values that are not marked as modifiable.
- Only PNO standard ID formats are supported.
- Additional parameters are not supported. Specifying additional parameters can lead to malfunctions.

GSD File
Fixed and Variable Settings

A GSD file describes the functional properties of a DP slave. Some parameters can be changed if required and some must be adapted. Others, on the other hand, have fixed values that must not be changed. The GSD file of the demonstration program is shown below and indicates the status of the settings.

Note that multiple module definitions are not permitted.

The configuration data are specified in bytes separated by commas and entered after the value module = "module1". Example: 1-byte input and 1-byte output appear as follows:

Module = "module1" 0x10, 0x20

The configuration data of the SOFTNET DP slave must not contain any empty slots (byte ID 00).
Additional DP Slave Information

#Profibus_DP
; Example: CP5511 slave with input and output data length of 8 bytes.
; Unit-Definition-List:
GSD_Revision=1 ; can be modified
Vendor_Name="SIEMENS AG" ; can be modified
Model_Name="SN DEMO Slave122b" ; must be modified
Revison="V1.0" ; can be modified
Ident_Number=0x8076 ; must be modified
Protocol_Ident=0 ; fixed
Station_Type=0 ; fixed
Hardware_Release="V2.0" ; can be modified
Software_Release="V1.0" ; can be modified
9.6_supp=1 ; supported transmission rates are fixed
19.2_supp=1 ; fixed
45.45_supp=1 ; fixed
93.75_supp=1 ; fixed
187.5_supp=1 ; fixed
500_supp=1 ; fixed
1.5M_supp=1 ; fixed
3M_supp=1 ; fixed
6M_supp=1 ; fixed
12M_supp=1 ; fixed

; fixed values dependent on transmission rate
MaxTsdr_9.6=60 ; fixed
MaxTsdr_19.2=60 ; fixed
MaxTsdr_45.45=400 ; fixed
MaxTsdr_93.75=60 ; fixed
MaxTsdr_187.5=60 ; fixed
MaxTsdr_500=100 ; fixed
MaxTsdr_1.5M=150 ; fixed
MaxTsdr_3M=250 ; fixed
MaxTsdr_6M=450 ; fixed
MaxTsdr_12M=800 ; fixed
Implementation_Type="ASPC2"; fixed
;Bitmap_Device="" ; can be modified

; Slave-Specification:
OrderNumber="6AV3 xx7-1xxxx-xxxx" ;must be modified
Auto_Baud_supp=0 ; fixed
Sync_Mode_supp=0 ; fixed
Freeze_Mode_supp=0 ; fixed
Min_Slave_Interval=20 ; fixed
Max_Diag_Data_Len=10 ; fixed
Modul_Offset=0 ; fixed
Slave_Family=3@TdF@B + B ; optional information. Example here: The slave
; appears in COM PROFIBUS in a catalog called B + B.;
Max_Module=1 ; fixed
Max_Input_Len=8 ; must be adapted. Possible values: 0 to 122
Max_Output_Len=8 ; must be adapted. Possible values: 0 to 122
; User_Prm_Data_Len=0 ; fixed

; Module-Definitions:
Module = "module1" 0x37
; Configuration : 8 bytes inputs/ 8 bytes outputs
; these bytes must be entered as cfg_data, see page 26
EndModule
5 Checklist for Programmers

Please note the following points when programming applications with the DP slave programming interface.

Program Structure
Make sure that you keep to the rules below when structuring your program:

- Start your application with a DPS_open and always complete it with a DPS_close.
- Remember that DP slave functions are synchronous. After the job is sent and completed, the return values are already available. Fetching the data later is not necessary.
- Only one job can be sent at any one time.
- Evaluate all reported errors.

Data Areas
Use the checklist below for your data areas:

- Select data buffer lengths to match the longest possible data (122 bytes).
- Use static data areas for your job fields and data areas.
- Link job fields with valid data buffers in the startup phase of your program.

Asynchronous Mode
Remember that jobs can only be sent to the DP slave lib from one thread of one application. The next job can only be sent after the previous job has been executed.
A Few Last Points

You should also keep in mind the following points:

- In your program, assume that “reserved” job and return parameters will be used in future for expansions. Assign a default value (0) to “reserved” elements.

- In bit-oriented return parameters, the reserved bits are also intended for future expansions. When checking these bit-oriented return parameters, mask reserved bits (for example by ANDing them with the value 0).

- Read the information in the FAQ for your product.

- Make sure that the lengths you specify for "DPS_set_input" or "DPS_get_output" match the values of your slave configuration for "DPS_open" and the values of your GSD file.

- Note the information about the startup behavior of the DP slave with the services DPS_start, DPS_set_input and DPS_get_output. The startup behavior of the DP slave may be different. This depends, for example, on whether the DP master or the DP slave was started first.

- The current state of the slave is transferred with the return value. No status query is necessary. If the slave is not in the DATA_EXCHANGE state, no data exchange with the DP master is possible. Return Value DPS_ERROR_EV_NO_DATA_EX (see Section 4.1.1).
6 Formats of the Slave Data

This chapter describes the formats of the slave data.

Contents of the Chapter

6.1 Formats of the Input and Output Data ............................................................... 44
6.2 Formats of the Slave Diagnostic Data ............................................................... 45
   6.2.1 Overview of the Entire Structure ............................................................... 46
   6.2.2 Format of the Diagnostic Data Header ....................................................... 47
6.1 Formats of the Input and Output Data

Order of the Slave Data
The order of the data corresponds to the configured input/output ports of the DP slaves. For example, the input ports of an ET 200B-16DI station are stored as follows: Port 0 in the first byte, Port 1 in the second byte, etc.

Format of Data Words
The following order is necessary for storing values in the word format (2-byte numbers):
First, the high byte (lower order address) of the word is entered followed by the low byte (higher order address).

Note that the SOFTNET DP slave supports only byte consistency.
6.2 Formats of the Slave Diagnostic Data

Introduction
In some cases, in the following data structures single bits in the bytes are significant. The numbering of the bits is always as follows: The least significant bit (LSB) has the number 0 and the most significant bit (MSB) has the number 7.

Note
The user program cannot access the diagnostic data.
6.2.1 Overview of the Entire Structure

Possible Length
The typical length of diagnostic data is between 6 and 32 bytes. The maximum possible length is 244 bytes.

Header and up to Three Following Fields
Diagnostic data have a fixed header and up to three further fields:

- **Header section**: Always present, fixed length *
- **Device-related diagnostics**: Optional, variable length
- **ID-related diagnostics**: Optional, variable length
- **Channel-related diagnostics**: Optional, variable length

* The diagnostic data of the CP 5511 slaves are restricted to the first 6 bytes of the header.
6.2.2 Format of the Diagnostic Data Header

Overall Structure of the Header

<table>
<thead>
<tr>
<th>byte</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 1</td>
<td>Stationstatus_1</td>
</tr>
<tr>
<td>Byte 2</td>
<td>Stationstatus_2</td>
</tr>
<tr>
<td>Byte 3</td>
<td>Stationstatus_3</td>
</tr>
<tr>
<td>Byte 4</td>
<td>Diag.Master_Add</td>
</tr>
<tr>
<td>Byte 5 and 6</td>
<td>Ident_Number</td>
</tr>
</tbody>
</table>
Each bit in byte 1, the "station_status_1" byte has a special meaning.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td><strong>Diag. Master_Lock</strong>, 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.</td>
</tr>
<tr>
<td>6</td>
<td><strong>Diag. Prm_Fault</strong>, This bit is set by the slave if the last parameter assignment frame was incorrect (for example wrong length, wrong ident number, invalid parameters).</td>
</tr>
<tr>
<td>5</td>
<td><strong>Diag. Invalid_Slave_Response</strong>, This bit is set as soon as an implausible response is received from an addressed DP slave.</td>
</tr>
<tr>
<td>4</td>
<td><strong>Diag. Not_Supported</strong>, This bit is set as soon as a function is requested that is not supported by the slave (for example operation in the SYNC mode although the slave does not support this).</td>
</tr>
<tr>
<td>3</td>
<td><strong>Diag. Ext_Diag</strong>, 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 user program.</td>
</tr>
<tr>
<td>2</td>
<td><strong>Diag. Cfg_Fault</strong>, 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.</td>
</tr>
<tr>
<td>1</td>
<td><strong>Diag. Station_Not_Ready</strong>, This bit is set when the DP slave is not yet ready for the productive phase.</td>
</tr>
<tr>
<td>0</td>
<td><strong>Diag. Station_Non_Existent</strong>, The DP master sets this bit, if the DP slave is not obtainable on 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.</td>
</tr>
</tbody>
</table>
Each bit in byte 2, the "station_status_2" byte has a special meaning.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td><strong>Diag. Deactivated</strong>, This bit is set as soon as the DP slave is indicated as inactive in the local parameter record and was taken out of cyclic processing.</td>
</tr>
<tr>
<td>6</td>
<td>reserved</td>
</tr>
<tr>
<td>5</td>
<td><strong>Diag. Sync_Mode</strong>, This bit is set by the DP slave as soon as it receives the Sync control command.</td>
</tr>
<tr>
<td>4</td>
<td><strong>Diag. Freeze_Mode</strong>, This bit is set by the DP slave as soon as it receives the Freeze control command.</td>
</tr>
<tr>
<td>3</td>
<td><strong>Diag. WD_On (Watchdog on)</strong>, This bit is set by the DP slave. If this bit is set to 1, the watchdog monitoring is activated on the DP slave.</td>
</tr>
<tr>
<td>2</td>
<td>The DP slave always sets this bit to 1.</td>
</tr>
<tr>
<td>1</td>
<td><strong>Diag.Stat_Diag (Static Diagnostics)</strong> - If the DP slave sets this bit, the DP master must fetch diagnostic information until the bit is cleared. The DP slave, for example, sets this bit when it is not capable of providing valid user data.</td>
</tr>
<tr>
<td>0</td>
<td><strong>Diag. Prm_Req</strong>, If the DP slave sets this bit, it must be re-assigned parameters and be reconfigured. This bit remains set until parameters have been assigned (if bit 1 and bit 0 are set, bit 0 has the higher priority).</td>
</tr>
</tbody>
</table>

Byte 3  
*(Stationstatus_3)*

<table>
<thead>
<tr>
<th>bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td><strong>Diag. Ext_Diag_Overflow</strong>, If this bit is set, there is more diagnostic data than can be indicated. 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.</td>
</tr>
<tr>
<td>6 to 0</td>
<td>reserved</td>
</tr>
</tbody>
</table>

Byte 4  
*(Diag. Master_Add)*

The address of the DP master that assigned parameters to the DP slave is entered here. If the DP slave has not been assigned parameters by a DP master, the DP slave sets the address 255 in this byte.
Bytes 5 and 6 (Ident_Number)  The vendor ID for a DP slave type is assigned in bytes 5 and 6, the "Ident_Number" byte. This identifier can be used on the one hand for test purposes and on the other for precise identification of the slave.
Glossary

COM PROFIBUS  Configuration tool for configuring PROFIBUS DP systems

CP  Communications Processor

CP 5511  The SOFTNET CPs:
- CP 5511 (PCMCIA card), CP5512 (PC-CARD card)
- CP 5611 (PCI card), CP 5611 A2 (PCI-X card)

CP 5614  Powerful CP with its own CPU and hardware-controlled DP prepro-
cessing.

DP  Distributed peripheral I/Os - fast, serial transmission protocol for con-
necting a central programmable controller with distributed I/O devices.

GSD file  Device database file (.GSD) contain DP slave descriptions complying
with EN 50170 or DIN E 19245 Part 3.

MCL1  Master class 1

MCL2  Master class 2

SIMATIC  Siemens automation system

SIMATIC NET  SIMATIC network architecture for automation and engineering - there
are three versions of this architecture:
- SIMATIC NET Industrial Ethernet
- SIMATIC NET PROFIBUS
- SIMATIC AS-i - bus at the actuator-sensor level

SSC  Interface Center in Fürth, Germany; certification center
# Index

<table>
<thead>
<tr>
<th>Diag. Cfg_Fault</th>
<th>48</th>
<th>Diag. Prm_Req</th>
<th>49</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diag. Deactivated</td>
<td>49</td>
<td>Diag. Station_Not_Ready</td>
<td>48</td>
</tr>
<tr>
<td>Diag. Ext_Diag</td>
<td>48</td>
<td>Diag. Stat_Diag</td>
<td>49</td>
</tr>
<tr>
<td>Diag. Ext_Diag_Overflow</td>
<td>49</td>
<td>Diag. Sync_Mode</td>
<td>49</td>
</tr>
<tr>
<td>Diag. Freeze_Mode</td>
<td>49</td>
<td>Diag. WD_On</td>
<td>49</td>
</tr>
<tr>
<td>Diag. Invalid_Slave_Response</td>
<td>48</td>
<td>Diag. Stat_Diag</td>
<td>49</td>
</tr>
<tr>
<td>Diag. Master_Add</td>
<td>50</td>
<td>Ident_Number</td>
<td>50</td>
</tr>
<tr>
<td>Diag. Master_Lock</td>
<td>48</td>
<td>Slave</td>
<td></td>
</tr>
<tr>
<td>Diag. Not_Supported</td>
<td>48</td>
<td>data format</td>
<td>44</td>
</tr>
<tr>
<td>Diag. Prm_Fault</td>
<td>48</td>
<td>format diagnostic data</td>
<td>45</td>
</tr>
</tbody>
</table>