SIEMENS

ASPC 2 / Software User Description

(Advanced PROFIBUS Controller
according to EN 50170)

Version: V1.1

Date: 01/15/2009

Liability Exclusion
We have tested the contents of this document regarding agreement with the hardware and software described. Nevertheless, there may be deviations, and we don't guarantee complete agreement. The data in the document is tested periodically, however. Required corrections are included in subsequent versions. We gratefully accept suggestions for improvement

Subject to technical changes.

Table of Contents

1 Introduction

The ASPC2 ASIC requires extensive software (i.e., approx. 64 Kbytes) for use as a DP master. Use of the software requires a license. Master license fees are 15 000€. for object code and 45 000€ for source code, regardless of the number desired.

2 Software Structure

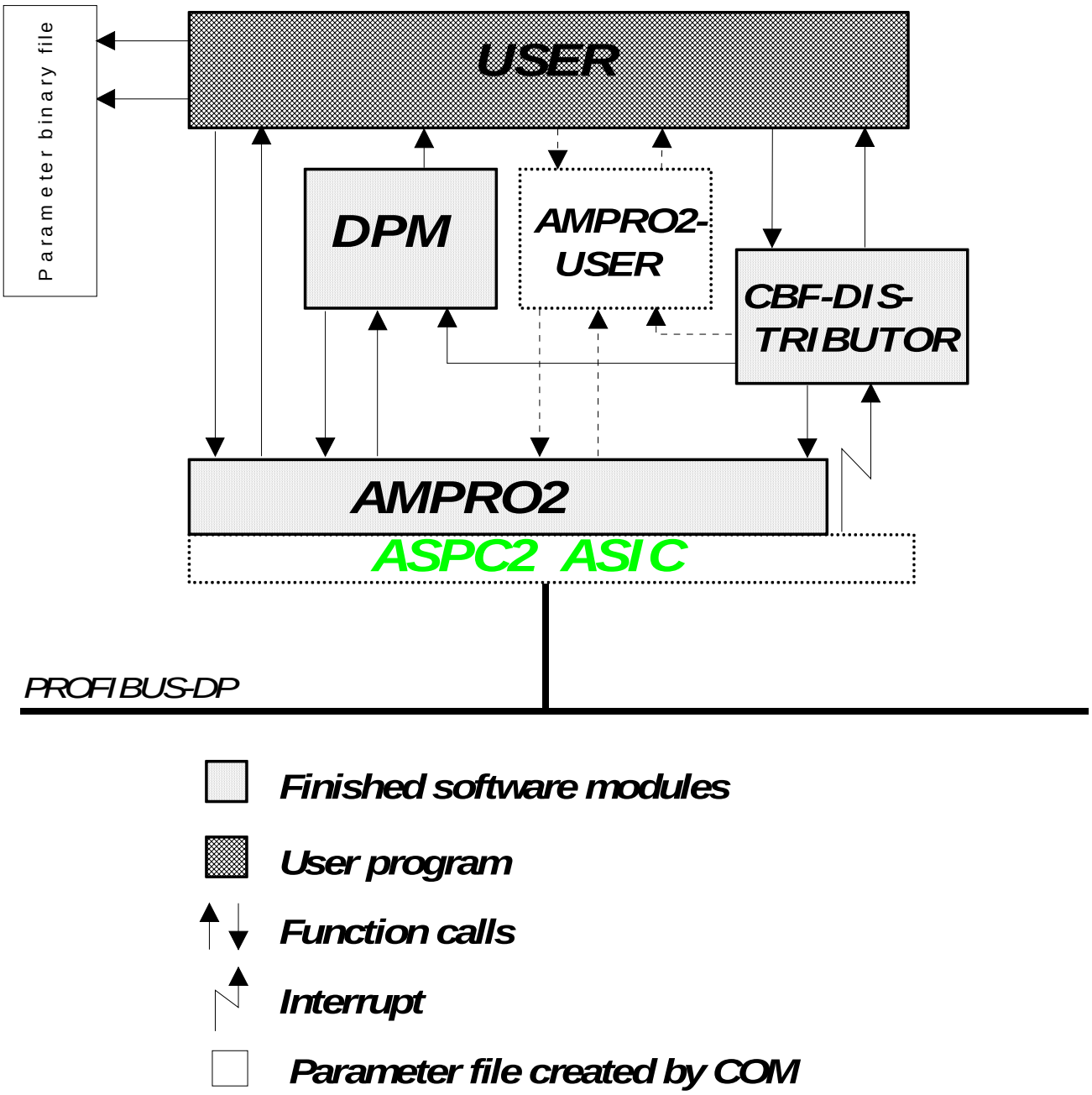


Figure 1: Software structure

2.1 Overview

2.1.1 Introduction

Since the software is designed to support a wide variety of compilers, processors, ASIC types, memory models and development environments, all function call interfaces and data pointers have been assigned attributes so that the finished software can be linked to the environment very precisely. It is important that the applicable header files have correct settings so that the finished modules can be optimally linked.

2.1.2 Delivery Form

AMPRO2, AMPRO-DPM and the CBF distributor are available in a library which the user can integrate in his programs, or with all source code files which are then generated by the user together with his own source code. How AMPRO2, AMPRO-DPM and the CBF distributor are delivered depends on the particular licensing contract (mandatory for both packages).

2.1.2.1 Library

When this form of delivery is selected, the sources must be generated with the type of compiler which the user would like to use before the sale takes place, since ANSI-C libraries do not exist.
This requires precise information on the settings for the different software modules and the development environment. See section 2.1.3.

2.1.2.2 C-Sources

When this form of delivery is selected, all sources of the software modules are included. These modules can be adapted to various development environments by using various settings.
See section 2.1.3.

2.1.3 Settings

Using various header files, the access attributes can be specified for the functions and data of the individual modules.
Header files are listed below.

File	Explanation
config.h	Setting for compiler type, ASIC, access attributes for AMPRO2 and DPM
config.txt	Detailed explanation of all possible settings in config.h
dev_def.h sys_comm.h sys_cbd.h	Access attributes for DPM and other AMPRO2 users Definitions for the integrated memory management of AMPRO2 Definitions and structures for the CBF

The sample application shown below explains most of the settings which you can made for these files.

2.1.3.1 Example

Prerequisite

Processor: 80C165
Development environment: BSO Tasking

2.1.3.1.1 config.h

The settings for the AMPRO2 module are made in the config.h file.

```
/*****************************************************************************/
/*          1.    POINTER-ATTRIBUTES                                         */
/*****************************************************************************/
/*---------------------------------------------------------------------------*/
/*          1.1   PTR_ATTR                                                   */
```

```
/*---------------------------------------------------------------------*/
#define PTR_ATTR_NEAR
/*---------------------------------------------------------------------*/
/*          1.2    L2_DATA_PTR_ATTR                                    */
/*---------------------------------------------------------------------*/
#define L2_DATA_PTR_ATTR_FAR
/*---------------------------------------------------------------------*/
/*          1.3    L2_CODE_ATTR                                        */
/*---------------------------------------------------------------------*/
#define L2_CODE_ATTR          ATTR_NEAR
/*---------------------------------------------------------------------*/
/*          1.4    L2_IFA_CODE_ATTR                                    */
/*---------------------------------------------------------------------*/
#define L2_IFA_CODE_ATTR      ATTR_NEAR
/*---------------------------------------------------------------------*/
/*          1.5    L2_CALL_BACK_CODE_ATTR                              */
/*---------------------------------------------------------------------*/
#define L2_CALL_BACK_CODE_ATTR  ATTR_NEAR
/*---------------------------------------------------------------------*/
/*          1.6    L2_STACK_DATA_ATTR                                  */
/*---------------------------------------------------------------------*/
/* not required for TOOL_CHAIN_TASKING */
/*---------------------------------------------------------------------*/
/*          1.7    L2_CLEAR_GLOBAL_DATA                                */
/*---------------------------------------------------------------------*/
/* not required for the IM 308-C */
/*********************************************************************/
/*          2.     L2-INTERFACE                                        */
/*********************************************************************/
#define L2_CALL_IFA_CALLBACK
/*********************************************************************/
/*          3.     HARDWARE                                            */
/*********************************************************************/
#define IM308C
#define L2_SHELL_CODE_ATTR    ATTR_HUGE
/*****************************************************************/
/*          4.     ASIC                                           */
/*****************************************************************/
/*---------------------------------------------------------------------*/
/*          4.1    SELECTION                                          */
/*---------------------------------------------------------------------*/
#define AMPRO2_ASPC2
/*---------------------------------------------------------------------*/
/*          4.2    ASPC2 selected                                     */
/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------*/
/*          4.2.1  ASPC2_ATTR                                         */
/*---------------------------------------------------------------------*/
#define ASPC2_ATTR_MEM_FAR
/*---------------------------------------------------------------------*/
/*          4.2.2  ASPC2_STEP                                         */
/*---------------------------------------------------------------------*/
#define ASPC2_GREATER_EQUAL_STEP_C
/*---------------------------------------------------------------------*/
/*          4.2.3  ASPC2_MEMORY-AREA                                  */
/*---------------------------------------------------------------------*/
#define L2_MEM_START_PAGE_HOST  0
/*---------------------------------------------------------------------*/
/*          4.2.4  ASPC2-ADR-MODE                                     */
/*---------------------------------------------------------------------*/
```



```
/* not used for the IM 308-C */
/*---------------------------------------------------------------------*/
/*            4.2.5   ASPC2-EOI                                        */
/*---------------------------------------------------------------------*/
/* not used for the IM 308-C */
/*---------------------------------------------------------------------*/
/*            4.2.6   L2-LOCK-MACROS                                   */
/*---------------------------------------------------------------------*/
/* not used for the IM 308-C */
/*---------------------------------------------------------------------*/
/*            4.2.7   L2-ASIC-LOCK                                     */
/*---------------------------------------------------------------------*/
/* not used for the IM 308-C */
/*************************************************************************/
/*            5.      HOST-uP                                          */
/*************************************************************************/
/*---------------------------------------------------------------------*/
/*            5.1     SELECTION                                        */
/*---------------------------------------------------------------------*/
#define PROC_80C165
/*---------------------------------------------------------------------*/
/*            5.3     80C165/80C167 selected                           */
/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------*/
/*            5.3.1   DATA-PAGE-LIMIT                                  */
/*---------------------------------------------------------------------*/
#define L2_MEM_DISABLE_DATA_PAGE_LIMIT
/*************************************************************************/
/*            6.      OPERATING-SYSTEM                                 */
/*************************************************************************/
/*---------------------------------------------------------------------*/
/*            6.1     SELECTION                                        */
/*---------------------------------------------------------------------*/
#define NO_OS
/*************************************************************************/
/*            7.      TOOL-CHAIN                                       */
/*************************************************************************/
#define TOOL_CHAIN_TASKING
#define L2_PRAGMA_GLOBAL
/*************************************************************************/
/*            8.      INTERNAL  (don't change !)                       */
/*************************************************************************/
#include "l2_attr.h"
```

2.1.3.1.2 dev_def.h

The dev_def.h file in accordance with the settings for the AMPRO2 user

```
/*                                                                     */
/*  D e s c r i p t i o n :                                            */
/*  This file includes all attribute-definitiones for one device. The  */
/*  USER has to decide, which kind of attributes he want's to use and  */
/*  than he has to edit the entries of this file according to his in-  */
/*  tention.                                                           */
/*                                                                     */
/*  A t t e n t i o n :                                                */
/*  This file must be created by the USER and he must keep all entries */
/*  in a good condition by himself according to the preconditions made */
/*  by any of the AMPRO-parts. If no USER-entries are required, the    */
/*  empty file-body must remain anyway. If AMPRO2s file CONFIG.H       */
/*  contains a define-value for the device, this define may be used to */
```

```
/*  protect the file against accidental usage. This technique is       */
/*  advantageous, but it's not essential.                              */
/*                                                                     */
/***********************************************************************/
/***********************************************************************/
/*+-------------------------------------------------------------------+*/
/*¦ Literallies                                                       ¦*/
/*+-------------------------------------------------------------------+*/
/* Reinclude-protection */
#ifndef COMM_DEV__DEV_DEF_H
#define COMM_DEV__DEV_DEF_H
/*+-------------------------------------------------------------------+*/
/*¦ Device-specific definitiones                                      ¦*/
/*+-------------------------------------------------------------------+*/
#if     defined (IM308C)    /* See COMMON\CONFIG.H for this definition. */
/*+-------------------------------------------------------------------+*/
   /*¦ AMPRO-parts used by the device                                  ¦*/
   /*+-----------------------------------------------------------------+*/
   /*
      With the following definitiones the USER has to tell AMPRO, which kind
      of functions his device is able to process. The four possible functions
      are DPM, DPS, DPX1 and DPX2. If any of the functions are needed, a
      definition must be created as follows:

         DEVICE_USES_XXXX    where XXXX mus be replacd by the components name.

      The entries should be enabled or disabled by using comment-signs and not
      by deleting any line.
   */
#define DEVICE_USES_DPM
#define DEVICE_USES_DPS

   /* #define DEVICE_USES_DPX1   */

   /* #define DEVICE_USES_DPX2   */
/*+-------------------------------------------------------------------+*/
   /*¦ Usage of AMPRO2-shell-functions                                 ¦*/
   /*+-----------------------------------------------------------------+*/
   /*
      With the following definitiones the USER has to tell AMPRO, if he want's
      to force any of the DP-components to use the AMPRO2-functions directly
      or to use shell-functions, so calling the AMPRO2-functions arcoss a
      second function. These shell functions are used to transform huge- or
      far-calls from one segment into near-calls inside another segment.
      The shell-functions may be selective enabled for each component enabled
      above. If any of the shell-function-set are needed, a definition must be
      created for each DP-component as follows:

         XXXX_USES_L2_SHELL  where XXXX mus be replacd by the components name.

      The entries should be enabled or disabled by using comment-signs and not
      by deleting any line.
   */
   #ifdef  DEVICE_USES_DPM
      /* #define DPM_USES_L2_SHELL */
   #endif

   #ifdef  DEVICE_USES_DPS
      #define DPS_USES_L2_SHELL
   #endif
```

```
#ifdef  DEVICE_USES_DPX1
   /* #define DPX1_USES_L2_SHELL */
#endif

#ifdef  DEVICE_USES_DPX2
   #define DPX2_USES_L2_SHELL
#endif


/*+-------------------------------------------------------------------------+*/
/*¦ area attributes                                                         ¦*/
/*+-------------------------------------------------------------------------+*/
/*
   The following attribute names must be changed according to the USERs
   hardware and locater rules. All attribute names consists of their
   primary name and the attribute in one definition. The attributes which
   can be used are ..._NEAR, ..._FAR and ..._HUGE. Further informations
   about each of the attribute names may be taken out of the specification
   for each DP-component.

   e.g.:
   The area HEINZ_DATA should have the attribute 'far', so it's attribute
   name listed below must look as follows:

      HEINZ_DATA_FAR
*/

/*- function attribute names ----------------------------------------------*/

#define DPM_IFA_FUNC_ATTR_HUGE         /* see AMPRO-DPM-specification  */
#define DPM_CALL_BACK_FUNC_ATTR_HUGE   /* see AMPRO-DPM-specification  */
#define DPM_INT_FUNC_ATTR_NEAR         /* see AMPRO-DPM-specification  */

#define DPX2_IFA_FUNC_ATTR_HUGE        /* see AMPRO-DPX2-specification */
#define DPX2_CALL_BACK_FUNC_ATTR_HUGE  /* see AMPRO-DPX2-specification */
#define DPX2_INT_FUNC_ATTR_NEAR        /* see AMPRO-DPX2-specification */

#define SYS_CBD_FUNC_ATTR_HUGE         /* attribute name of the main CBF-server */

#define EXT_FUNC_ATTR_NEAR             /* old attribute name */


/*- data attribute names --------------------------------------------------*/

#define DPM_IFA_DATA_ATTR_NEAR         /* see AMPRO-DPM-specification  */
#define DPM_INT_DATA_ATTR_NEAR         /* see AMPRO-DPM-specification  */

#define DPX2_INT_DATA_ATTR_NEAR        /* see AMPRO-DPX2-specification */
#define DPX2_IFA_DATA_ATTR_NEAR        /* see AMPRO-DPX2-specification */

#define DPM_USER_DATA_ATTR_NEAR        /* see AMPRO-DPM-specification  */
#define DPM_PROC_DATA_ATTR_FAR         /* see AMPRO-DPM-specification  */
#define DPM_MOD_DATA_ATTR_HUGE         /* see AMPRO-DPM-specification  */

#define DPX2_USER_DATA_ATTR_NEAR       /* see AMPRO-DPX2-specification */
#define DPX2_PROC_DATA_ATTR_FAR        /* see AMPRO-DPX2-specification */
#define DPX2_MOD_DATA_ATTR_HUGE        /* see AMPRO-DPX2-specification */

#define ERR_DATA_ATTR_NEAR             /* attribute name for ERROR_CB  */
```

```
   #define PAM_MOD_ATTR_HUGE          /* attribute name for modul PAM */
   #define DAM_DPR_ATTR_FAR           /* attribute name for modul DAM */
   #define DAM_ZRAM_ATTR_HUGE         /* attribute name for modul DAM */

   #define S7_DATA_ATTR_NEAR          /* attribute name for modul DAM */

   #define EXT_DATA_ATTR_NEAR         /* old attribute name */


   /*+-------------------------------------------------------------------+*/
   /*¦ Further Definitiones                                              ¦*/
   /*+-------------------------------------------------------------------+*/

   /*- module format ----------------------------------------------------*/
   /*
      The parameter module data structures may have high-endian- (Intel-) or
      low-endian- (Motorola-) format. The following define value tells AMPRO-DPM
      the format type of the parameter module. Two different values are possible:

      high-endian-format:    DPM_MODULE_FORMAT_HIGH_ENDIAN   (Intel)
      low-endian-format:     DPM_MODULE_FORMAT_LITTLE_ENDIAN (Motorola)
   */

   #define DPM_MODULE_FORMAT_HIGH_ENDIAN
   #define DPX2_MODULE_FORMAT_HIGH_ENDIAN


#else

   #error "The file COMM_DEV\DEV_INC.H was created only for the device IM 308-C !"

#endif



/*+-------------------------------------------------------------------------+*/
/*¦ Literallies                                                             ¦*/
/*+-------------------------------------------------------------------------+*/

/* Reinclude-protection */
#else
   #pragma message ("The header COMM_DEV\DEV_DEF.H is included twice or more !")
#endif
```

2.1.3.1.3 sys_comm.h

The sys_comm.h file contains the settings for memory management.

```
/*****************************************************************************/
/*****************************************************************************/
/*+-------------------------------------------------------------------------+*/
/*¦ Literallies                                                             ¦*/
/*+-------------------------------------------------------------------------+*/
/* Reinclude-protection */
#ifndef COMM_DEV__SYS_COMM_H
#define COMM_DEV__SYS_COMM_H
/*+-------------------------------------------------------------------------+*/
/*¦ SYS_CBD-Handling for ASIC Interrupts                                    ¦*/
```



```
/*+-------------------------------------------------------------------------+*/
/*
   Interrupt Processing
   There are three possibilities to process the AMPRO2-interrupts:

   If the following macro is defined, the ASIC-queues are emptied full.
   This means: The next ASIC Interrupt will be generated after execution of
         all con/ind list entries or if the 1ms timer is expired.

   #define SYS_EMPTY_ASIC_QUEUES_FULL

   If the following macro is defined, the ASIC-queues are emptied single.
   This means: The next ASIC Interrupt will be generated earliest after 1ms.
         Every interrupt only one con/ind list entry will be executed.

   #define SYS_EMPTY_ASIC_QUEUES_SINGLE

   If the following macros are defined, the ASIC-queues are emptied as often
   as the given number shows.
   This means: The next ASIC Interrupt will be generated after execution of
         up to 'cnt' con/ind list entries or if the 1ms timer is expired.

   #define SYS_EMPTY_ASIC_QUEUES_MULTIPLE
   #define SYS_EMPTY_ASIC_QUEUES_CNT      (UBYTE) 0xNN
*/

#define SYS_EMPTY_ASIC_QUEUES_MULTIPLE
#define SYS_EMPTY_ASIC_QUEUES_CNT         (UBYTE) 0x02


#if !defined(SYS_EMPTY_ASIC_QUEUES_FULL) && !defined(SYS_EMPTY_ASIC_QUEUES_SINGLE) &&
!defined(SYS_EMPTY_ASIC_QUEUES_MULTIPLE)
   #pragma error ("Within COMM_DEV\SYS_COMM.H is no AMPRO2-Int processing technique defined !")
#endif

#if                              (         defined(SYS_EMPTY_ASIC_QUEUES_FULL)                    &&
defined(SYS_EMPTY_ASIC_QUEUES_MULTIPLE) )
   || ( defined(SYS_EMPTY_ASIC_QUEUES_FULL)   && defined(SYS_EMPTY_ASIC_QUEUES_SINGLE)
)
   ||                  (                  defined(SYS_EMPTY_ASIC_QUEUES_SINGLE)                  &&
defined(SYS_EMPTY_ASIC_QUEUES_MULTIPLE) )
   #pragma error ("Within COMM_DEV\SYS_COMM.H is only one AMPRO2-Int processing technique
allowed !")
#endif

#if   ( defined(SYS_EMPTY_ASIC_QUEUES_FULL )  && defined(SYS_EMPTY_ASIC_QUEUES_CNT)  )
   || ( defined(SYS_EMPTY_ASIC_QUEUES_SINGLE)  && defined(SYS_EMPTY_ASIC_QUEUES_CNT)  )
   || ( defined(SYS_EMPTY_ASIC_QUEUES_MULTIPLE) && !defined(SYS_EMPTY_ASIC_QUEUES_CNT)
)
   #pragma error ("Within COMM_DEV\SYS_COMM.H only with SYS_EMPTY_ASIC_QUEUES_MULTIPLE
SYS_EMPTY_ASIC_QUEUES_CNT is allowed !")
#endif



/*+-------------------------------------------------------------------------+*/
/*¦ AMPRO2-Memory-Management (all lens are in byte !)  ¦*/
/*+-------------------------------------------------------------------------+*/

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

```
/*¦ Application-Blocks                                                    ¦*/
/*+-------------------------------------------------------------------------+*/

/*- MCP ---------------------------------------------------------------------*/

   #define SYS_MCP_L2_APB_NO          1    /* used for AMPRO2-FLC...-Reset as well as for ASIC-WD */
   #define SYS_MCP_L4_APB_NO          0


/*- AMPRO-DPM ----------------------------------------------------------------*/
/*
   The following application-blocks are used by DPM:

   L2-APBs fixed:     3 set_master_mode:     exchange, lock, withdraw
                +1 mark_cycle:          mark
                +1 init:                bus_access
                +1 set_slave_address:   ssla
                +8 set_slave_mode:      dummy (depending on DPM_MAX_GC_REQ_NO)
                ---------------------------------------------------------
                14 L2-APBs fixed

   L2-APBs per slave:  2 input/output_update: change_i_buffer, change_o_buffer
                +1                      diag
                +1                      data
                +1 withdraw_slave:      withdraw_repeat
                +1                      cfg, not used for DP-Siemens-SPM-Slaves
                +1 set_master_mode:     clear, only used for DP-Siemens-Slaves
                ---------------------------------------------------------
                 7 L2-APBs per slave

   L4-APBs fixed:     1 timer_expired:   tex
                +1 set_master_mode: smm
                +8 set_slave_mode:  ssm (depending on DPM_MAX_GC_REQ_NO)
                ---------------------------------------------------
                10 L4-APBs fixed

   L4-APBs per slave:  1 prm
                ------------------------------
                 1 L4-APBs per slave
*/

#ifdef DEVICE_USES_DPM
   #define SYS_DPM_MAX_SLAVE_NO       122        /* max. number of slaves  */

   #define SYS_DPM_L2_APB_NO_FIX      (6 + DPM_MAX_GC_REQ_NO)
   #define SYS_DPM_L2_APB_NO_VAR       7
   #define SYS_DPM_L2_APB_NO          ( SYS_DPM_L2_APB_NO_FIX + (SYS_DPM_MAX_SLAVE_NO *
SYS_DPM_L2_APB_NO_VAR) )

   #define SYS_DPM_L4_APB_NO_FIX      (2 + DPM_MAX_GC_REQ_NO)
   #define SYS_DPM_L4_APB_NO_VAR       1
   #define SYS_DPM_L4_APB_NO          ( SYS_DPM_L4_APB_NO_FIX + (SYS_DPM_MAX_SLAVE_NO *
SYS_DPM_L4_APB_NO_VAR) )
#else
   #define SYS_DPM_L2_APB_NO          0
   #define SYS_DPM_L4_APB_NO          0
#endif


/*- AMPRO-DPS ----------------------------------------------------------------*/
```

```
#ifdef DEVICE_USES_DPS
   #define SYS_DPS_L2_APB_NO            41     /* 1 used by MCP for L2_MAC_RESET in case of DPS active. */
                                /* 1 APB added for ASPC2 error workaround. StPo/04.07.96 */
   #define SYS_DPS_L4_APB_NO        0
#else
   #define SYS_DPS_L2_APB_NO        0
   #define SYS_DPS_L4_APB_NO        0
#endif


/*- AMPRO-DPX1 ---------------------------------------------------------------*/

#ifdef DEVICE_USES_DPX1
   #define SYS_DPX1_L2_APB_NO       0
   #define SYS_DPX1_L4_APB_NO       0
#else
   #define SYS_DPX1_L2_APB_NO       0
   #define SYS_DPX1_L4_APB_NO       0
#endif


/*- AMPRO-DPX2 ---------------------------------------------------------------*/

#ifdef DEVICE_USES_DPX2
   #define SYS_DPX2_L2_APB_NO       4
   #define SYS_DPX2_L4_APB_NO       3
#else
   #define SYS_DPX2_L2_APB_NO       0
   #define SYS_DPX2_L4_APB_NO       0
#endif


/*- Defines for all components -----------------------------------------------*/

#define SYS_L2_APB_PRE_RESERVED      0
#define SYS_L2_APB_POST_RESERVED      0

#define SYS_L2_APB1_PRE_RESERVED     SYS_L2_APB_PRE_RESERVED
#define SYS_L2_APB1_POST_RESERVED    SYS_L2_APB_POST_RESERVED


#define SYS_L4_APB_PRE_RESERVED      0
#define SYS_L4_APB_POST_RESERVED     DPM_L4_APB_DATA_LEN    /* = l4-header-length, must be even !!! */

#define SYS_L2_APB2_PRE_RESERVED     SYS_L4_APB_PRE_RESERVED
#define SYS_L2_APB2_POST_RESERVED    SYS_L4_APB_POST_RESERVED


#define SYS_L2_APB_NO            ( SYS_MCP_L2_APB_NO   \
                         + SYS_DPM_L2_APB_NO   \
                         + SYS_DPS_L2_APB_NO   \
                         + SYS_DPX1_L2_APB_NO  \
                         + SYS_DPX2_L2_APB_NO)
#define SYS_L2_APB1_NR           SYS_L2_APB_NO

#define SYS_L4_APB_NO            ( SYS_MCP_L4_APB_NO   \
                         + SYS_DPM_L4_APB_NO   \
```

```
                        + SYS_DPS_L4_APB_NO    \
                        + SYS_DPX1_L4_APB_NO   \
                        + SYS_DPX2_L4_APB_NO)
#define SYS_L2_APB2_NR              SYS_L4_APB_NO


#define SYS_LEN_MEM_L2_APB          (SYS_L2_APB_NO * (  (correct_size_to_even__ (L2_LEN_APB))
\
                                  + (correct_size_to_even__ (SYS_L2_APB_PRE_RESERVED))  \
                                  + (correct_size_to_even__ (SYS_L2_APB_POST_RESERVED))))
#define SYS_L2_LEN_MEM_APB1         SYS_LEN_MEM_L2_APB

#define SYS_LEN_MEM_L4_APB          (SYS_L4_APB_NO * (  (correct_size_to_even__ (L2_LEN_APB))
\
                                  + (correct_size_to_even__ (SYS_L4_APB_PRE_RESERVED))  \
                                  + (correct_size_to_even__ (SYS_L4_APB_POST_RESERVED))))
#define SYS_L2_LEN_MEM_APB2         SYS_LEN_MEM_L4_APB


#define SYS_LEN_MEM_APB             (SYS_LEN_MEM_L2_APB + SYS_LEN_MEM_L4_APB)



/*+-------------------------------------------------------------------------+*/
/*¦ Data-Blocks                                                             ¦*/
/*+-------------------------------------------------------------------------+*/

/*- AMPRO-DPM ---------------------------------------------------------------*/



/*- AMPRO-DPS ---------------------------------------------------------------*/

#ifdef DEVICE_USES_DPS
   #define SYS_L2_DB4_NR_OVERHEAD_DPS   49        /* number of db4  for DPS2 */
#endif



/*- AMPRO-DPX1 --------------------------------------------------------------*/


/*- AMPRO-DPX2 --------------------------------------------------------------*/

#ifdef DEVICE_USES_DPX2
   #define SYS_L2_DB4_NR_OVERHEAD_DPX2   2        /* number of db4  for DPX2 */
#endif


/*- Defines for all components ----------------------------------------------*/

#define SYS_L2_DB_PRE_RESERVED          0
#define SYS_L2_DB_POST_RESERVED         0

#define SYS_L2_MAC_EVENT_BUF_NR         3       /* mac-event-buffer */
#define SYS_L2_MAC_EVENT_BUF_SIZE       200

#define SYS_L2_LAS_EVENT_BUF_NR         3       /* las-event-buffer */
#define SYS_L2_LAS_EVENT_BUF_SIZE       128       /* was (HSA + 1) */
```

```
#define SYS_L2_LEN_SCB_RESERVE          0x10      /* SCB should be segment-aligned (Microsoft: base_seg = 16 Bit) */
#define SYS_L2_LEN_MEM_RESERVE          1     /* memory-block should be word-aligned -> reserve = 1 Byte */

/* length of the APM_VAR-area ca. 22k (0x5800): 16k APB1, 4k APB2, 2k rest (SCB, Init-Blk, BusPar-Blk, Event-Bufs (MAC, LAS) */
#define SYS_L2_LEN_MEM_VAR          (                   (correct_size_to_even__ (L2_LEN_SCB)) \
                                        + (correct_size_to_even__ (L2_LEN_INIT_BLOCK))      \
                                        + (correct_size_to_even__ (L2_LEN_BUS_PAR_BLOCK))   \
                        +       SYS_L2_MAC_EVENT_BUF_NR     *       (correct_size_to_even__ (SYS_L2_MAC_EVENT_BUF_SIZE))  \
                        +       SYS_L2_LAS_EVENT_BUF_NR     *       (correct_size_to_even__ (SYS_L2_LAS_EVENT_BUF_SIZE))  \
                                        + (correct_size_to_even__ (SYS_L2_LEN_SCB_RESERVE))     \
                                        + (correct_size_to_even__ (SYS_L2_LEN_MEM_RESERVE)) )

#define SYS_L2_LEN_MEM_DATA     0x20000     /* 128k for data-blocks */

#define SYS_L2_LEN_MEM_APB_VAR      (SYS_LEN_MEM_APB + SYS_L2_LEN_MEM_VAR)



/*+-------------------------------------------------------------------------+*/
/*¦ structures / variables / definitions                                    ¦*/
/*+-------------------------------------------------------------------------+*/

/*+-------------------------------------------------------------------------+*/
/*¦ Variables                                                               ¦*/
/*+-------------------------------------------------------------------------+*/

/*- AMPRO2-Memory-Management ------------------------------------------------*/

#if defined (SYS)

   #pragma save_attributes
   #pragma global
   #pragma class   NB=SYS_L2_MEM_APB
   #pragma combine NB=P

        ubyte       sys_l2_mem_apb_var  [SYS_L2_LEN_MEM_APB_VAR];

   #pragma class   HB=SYS_L2_MEM_DATA
   #pragma combine HB=P

        ubyte   huge    sys_l2_mem_data     [SYS_L2_LEN_MEM_DATA];

   #pragma restore_attributes

#else

   extern  ubyte       sys_l2_mem_apb_var  [SYS_L2_LEN_MEM_APB_VAR];
   extern  ubyte   huge    sys_l2_mem_data     [SYS_L2_LEN_MEM_DATA];

#endif



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

```
/*¦ Function Prototypes                                                    ¦*/
/*+-------------------------------------------------------------------------+*/

#if defined (SYS)

   #pragma save_attributes
   #pragma global

        void    huge    system_error_function   (ERRCB);

   #pragma restore_attributes

#else

   extern  void    huge    system_error_function   (ERRCB);

#endif



/*+-------------------------------------------------------------------------+*/
/*¦ Literallies                                                            ¦*/
/*+-------------------------------------------------------------------------+*/

/* Reinclude-protection */
#else
   #pragma message ("The header COMM_DEV\SYS_COMM.H is included twice or more !")
#endif
```

2.1.3.1.4 sys_cbd.h

```
/*+-------------------------------------------------------------------------+*/
/*¦ Literallies                                                            ¦*/
/*+-------------------------------------------------------------------------+*/

/* Reinclude-protection */
#ifndef COMM_DEV__SYS_CBD_H
#define COMM_DEV__SYS_CBD_H



/*+-------------------------------------------------------------------------+*/
/*¦ CBD: Definitiones                                                      ¦*/
/*+-------------------------------------------------------------------------+*/

#if defined SYS_CBD

   /*+-------------------------------------------------------------------------+*/
   /*¦ Code-Pragmas                                                           ¦*/
   /*+-------------------------------------------------------------------------+*/

   #pragma class   PR=SYS_CB_SERVER_CODE
   #pragma combine PR=P


   /*+-------------------------------------------------------------------------+*/
   /*¦ cbf_server_function quantities                                         ¦*/
   /*+-------------------------------------------------------------------------+*/
   /*
      Attention please: For the IM308C-Hardware
```



```
        - the FUNC1-define is used by the dpm_l2_cb_server,
        - the FUNC2-define is used by the dpx1_l2_cb_server,
        - the FUNC3-define is used by the dpx2_l2_cb_server,
        - the FUNC4-define is used by the dps_l2_cb_server.

      This general layout can be used by every AMPRO2-CBF-USER.
   */

   #define SYS_CBD_FUNC1_CBF_NR    70 /* numbers of cbfs for cbf-server-function1 */
   #define SYS_CBD_FUNC2_CBF_NR    20 /* numbers of cbfs for cbf-server-function2 */
   #define SYS_CBD_FUNC3_CBF_NR    20 /* numbers of cbfs for cbf-server-function3 */
   #define SYS_CBD_FUNC4_CBF_NR    20 /* numbers of cbfs for cbf-server-function4 */


   /* The following values are only used with the APB's element 'subsystem', */
   /* therefore they are directly casted to AMPRO2's type UBYTE. */

   #define SYS_CBD_FUNC1_MIN_NR    (UBYTE) 0   /* function1 begins with subsystem-no. 0 */
   #define      SYS_CBD_FUNC1_MAX_NR                    (UBYTE)      (SYS_CBD_FUNC1_MIN_NR      +
SYS_CBD_FUNC1_CBF_NR - 1)

   #define SYS_CBD_FUNC2_MIN_NR    (UBYTE) (SYS_CBD_FUNC1_MAX_NR + 1)
   #define      SYS_CBD_FUNC2_MAX_NR                    (UBYTE)      (SYS_CBD_FUNC2_MIN_NR      +
SYS_CBD_FUNC2_CBF_NR - 1)

   #define SYS_CBD_FUNC3_MIN_NR    (UBYTE) (SYS_CBD_FUNC2_MAX_NR + 1)
   #define      SYS_CBD_FUNC3_MAX_NR                    (UBYTE)      (SYS_CBD_FUNC3_MIN_NR      +
SYS_CBD_FUNC3_CBF_NR - 1)

   #define SYS_CBD_FUNC4_MIN_NR    (UBYTE) (SYS_CBD_FUNC3_MAX_NR + 1)
   #define      SYS_CBD_FUNC4_MAX_NR                    (UBYTE)      (SYS_CBD_FUNC4_MIN_NR      +
SYS_CBD_FUNC4_CBF_NR - 1)

#endif


/*+-------------------------------------------------------------------------+*/
/*¦ Structures                                                              ¦*/
/*+-------------------------------------------------------------------------+*/

struct  sys_cbd_def
{
   /* CBF-server-function pointers */
   void    func_ptr_name__(SYS_CBD_FUNC_ATTR, error)             (ERRCB);
   void    func_ptr_name__(SYS_CBD_FUNC_ATTR, server_function_1)   (L2_APB_PTR);
   void    func_ptr_name__(SYS_CBD_FUNC_ATTR, server_function_2)   (L2_APB_PTR);
   void    func_ptr_name__(SYS_CBD_FUNC_ATTR, server_function_3)   (L2_APB_PTR);
   void    func_ptr_name__(SYS_CBD_FUNC_ATTR, server_function_4)   (L2_APB_PTR);

};



/*+-------------------------------------------------------------------------+*/
/*¦ Variables, External-Function-Definitiones                               ¦*/
/*+-------------------------------------------------------------------------+*/

#if defined SYS_CBD

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

```
   /*¦ CBD: Variables                                                    ¦*/
   /*+-------------------------------------------------------------------+*/

   #pragma save_attributes
   #pragma class                NB=SYS_CB_SERVER_DATA
   #pragma combine                NB=P

      struct  sys_cbd_def _sys_cbd;

   #pragma restore_attributes


   /*+-------------------------------------------------------------------+*/
   /*¦ CBD: External-Function-Definitiones                               ¦*/
   /*+-------------------------------------------------------------------+*/

   extern  void    SYS_CBD_FUNC_ATTR   server_function_1   (L2_APB_PTR);
   extern  void    SYS_CBD_FUNC_ATTR   server_function_2   (L2_APB_PTR);
   extern  void    SYS_CBD_FUNC_ATTR   server_function_3   (L2_APB_PTR);
   extern  void    SYS_CBD_FUNC_ATTR   server_function_4   (L2_APB_PTR);

#else

   /*+-------------------------------------------------------------------+*/
   /*¦ Variables for common use                                          ¦*/
   /*+-------------------------------------------------------------------+*/

   extern  struct  sys_cbd_def _sys_cbd;

#endif



/*+-------------------------------------------------------------------+*/
/*¦ Literallies                                                       ¦*/
/*+-------------------------------------------------------------------+*/

/* Reinclude-protection */
#else
   #pragma message ("The header COMM_DEV\SYS_CBD.H is included twice or more !")
#endif
```

2.1.4 Procedure

The following must be included in the user program.

- ☞ Initialization and any allocation of memory blocks for AMPRO2 and DPM
- ☞ Initialize the CBF distributor and include in the interrupt routine of the processor
- ☞ Set up structure for the FLC_FMA_MAC_RESET reset function of AMPRO2 and execute the function
- ☞ Communication with DPM via the functions and call back functions

2.1.5 Scope of Delivery

☞ Files in path SRC_DIR\COMMON:

File	Explanation
ampro2.h	Conditional include calls depending on the processor and ASIC

config.fra	Fragment for config.h
config.h	Setting for compiler type, ASIC, access attributes for AMPRO2 and DPM
config.txt	Detailed explanation of all possible settings of config.h
dpm_comm.h	Definitions and structures for DPM
dpm_err.h	Definitions for error function for DPM
dp_attr.h	Attributes for function calls and memory accesses for the various modules
dp_comm.h	Conditional include calls depending on the modules used
dp_defma.h	Macros and attributes for function calls and memory accesses
dp_error.h	General definitions for error function and appropriate include calls
dp_types.h	General definitions
dp_vers.txt	Version of the global header files
hstruktr.txt	Structure of the linked head files
im308c.cfg	Configuration for IM308C
l2_attr.h	Attributes for memory access and function calls depending on the settings
l2_user.h	Definitions and structures for AMPRO2
l2_util.h	ASPC2 macros
plausibl.h	Check settings in config.h
typ_asp2.h	Definitions and structures for ASPC2
typ_comp.h	General definitions

☞ Files in path SRC_DIR\COMM_DEV:

File	Explanation
ap2_err.h	Definitions for error function for AMPRO2
dev_def.h	Access attributes for DPM and other AMPRO2 users
dev_err.h	Include calls for error function for general modules
dev_inc.h	Include calls for general modules
haw_err.h	Definitions for error function for hardware module
im_types.h	General definitions
sys_cbd.h	Definitions and structures for the CBF distributor
sys_comm.h	Definitions for the integrated memory management of AMPRO2
sys_err.h	Definitions for error function
sys_shll.h	Definitions for the shell functions for memory management

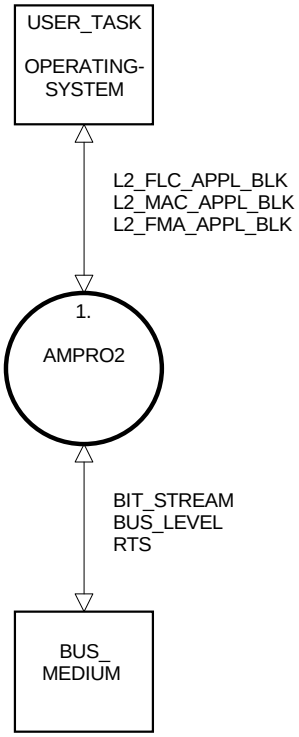
☞ Files in path SRC_DIR\AMPRO2\COMMON:

File	Explanation
ampro2.h	Conditional include calls depending on the processor and the ASIC
config.fra	Fragment for config.h
config.h	Setting for compiler type, ASIC, access attributes for AMPRO2 and DPM
config.txt	Detailed explanation of all possible settings of config.h
l2_attr.h	Attributes for memory access and function calls depending on the settings
l2_user.h	Definitions and structures for AMPRO2
l2_util.h	ASPC2 macros
plausibl.h	Check settings in config.h
typ_asp2.h	Definitions and structures for ASPC2
typ_comp.h	General definitions

☞ Applicable sources or library files for the individual modules

3 AMPRO2 Interface

- Basis for **AMPRO2** (**A**dvanced **M**ulti-user **PRO**fibus layer **2)** is PROFIBUS standard EN 50170, part 1.



3.1 User Interface

- **AMPRO2** (**A**dvanced **M**ulti-user **PRO**fibus layer **2)** is the link between BUS_MEDIUM and the LAYER2 user (e.g., AMPRO4).

- To issue a job (**REQ**uest), the user must supply a structured memory area for management and user data. So-called application blocks are defined for the various classes of services.

L2_FLC_APPL_BLK	For FLC services
L2_MAC_APPL_BLK	For MAC services
L2_FMA_APPL_BLK	For FMA services

  A job can be issued by the user (USER_TASK) via an operating system call (OPERATING_SYSTEM) or via a direct AMPRO2 call.

- Confirmation of execution (**CON**firmation) or indication of a received telegram (**IND**ication) or an event (**FMA_IND**ication) is also performed with an application block.

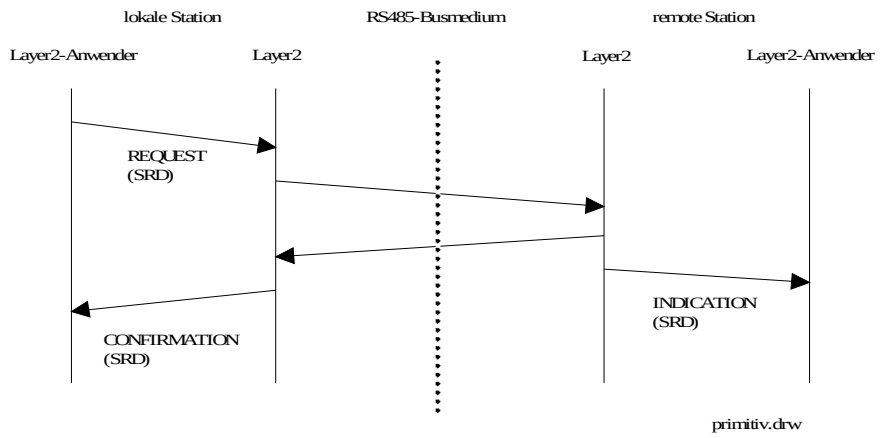
  This is returned to the mailbox of the user when the operating system environment is used. When an operating system environment is not used, this can be fetched with an AMPRO2 call.

The scope of performance of the user interface is described in detail in the next few sections. Since this user interface applies to all AMPRO2 implementations, it should be pointed out that each implementation does not offer all services described. Each service contains a remark to this effect.
The description of this interface requires a knowledge of the PROFIBUS standard.

3.2 Service Primitives

- **REQUEST**
  This primitive is used to transfer layer-2 service requests of a layer-2 user. A distinction is made between FMA requests, MAC requests and FLC requests depending on the type of service.
- **CONFIRMATION**
  This primitive is used to confirm to the user the request after performance of the service by layer 2. Depending on the service performed, FMA confirmation, MAC confirmation or FLC confirmation is then provided.
- **INDICATION**
  This primitive is used to report events to the layer-2 user. A distinction is made between telegram receipt (MAC indication) and other events in the MAC (FMA indication) or FLC (FLC indication).

The following diagram shows an example of MAC service SRD for better comprehension of the location of the service primitives from the viewpoint of the layer-2 user.



Detailed list of all service primitives

(0x15)	**L2_FLC_CONFIRM_INVALID_APB**	Acknowledgment of a service request with an application block from an invalid memory area (only occurs with AMPRO2-ASPC2 with *PTR_ATTR_FAR*)
(0x10)	**L2_FLC_CONFIRM_INVALID_OPCODE**	Acknowledgment of a service request with an invalid job (invalid *opcode* in L2 application block)
(0x10)	**L2_FMA_REQUEST**	Service request for an FMA job
(0x0D)	**L2_FMA_CONFIRM**	Acknowledgment of an FMA job
(0x11)	**L2_FMA_INDICATION**	Reporting of events in the MAC
(0x04)	**L2_MAC_REQUEST_LOW**	Service request for a low-priority MAC job
(0x05)	**L2_MAC_REQUEST_HIGH**	Service request for a high-priority MAC job
(0x06)	**L2_MAC_CONFIRM_LOW**	Acknowledgment of a low-priority MAC job
(0x07)	**L2_MAC_CONFIRM_HIGH**	Acknowledgment of a high-priority MAC job
(0x02)	**L2_MAC_INDICATION_LOW**	Reporting of receipt of a low-priority telegram in MAC
(0x03)	**L2_MAC_INDICATION_HIGH**	Reporting of receipt of a high-priority telegram in MAC
(0x09)	**L2_MAC_REP_REQUEST**	Service request for a MAC repeat job
(0x0E)	**L2_MAC_REP_CONFIRM_NEGATIVE**	Return of a MAC repeat job when an unexpected response occurs
(0x0F)	**L2_MAC_REP_CONFIRM_WITHDRAW**	Return of a MAC repeat job after MAC_REPEAT_WITHDRAW request
(0x18)	**L2_REP_REQUEST**	Service request for a MAC repeat auxiliary job
(0x14)	**L2_REP_CONFIRM**	Acknowledgment of a MAC repeat auxiliary job
(0x14)	**L2_MAC_REQUEST_COLLECT_LOW**	Service request for a low-priority MAC group job
(0x15)	**L2_MAC_REQUEST_COLLECT_HIGH**	Service request for a high-priority MAC group job
(0x16)	**L2_MAC_CONFIRM_COLLECT_LOW**	Acknowledgment of a low-priority MAC group job
(0x17)	**L2_MAC_CONFIRM_COLLECT_HIGH**	Acknowledgment of a high-priority MAC group job
(0x19)	**L2_MAC_REQUEST_DP_COLLECT**	Service request for a MAC-DP group job
(0x1A)	**L2_MAC_CONFIRM_DP_COLLECT**	Acknowledgment of a MAC-DP group job
(0x1C)	**L2_MAC_REQUEST_DP_COLLECT_FPT**	Service request for a MAC-DP group job with "Force Pass-Token"
(0x1D)	**L2_MAC_CONFIRM_DP_COLLECT_FPT**	Acknowledgment of a MAC-DP group job with "Force Pass-Token"
(0x00)	**L2_FLC_REQUEST**	Service request for an FLC job
(0x01)	**L2_FLC_CONFIRM**	Acknowledgment of an FLC job
(0x0A)	**L2_FLC_INDICATION**	Return of an old SRD response buffer
(0x12)	**L2_FLC_IND_RES_WITHDRAW**	Return of an indication resource after IND_RESOURCE_WITHDRAW request
(0x1A)	**L2_FLC_IND_REPLY_WITHDRAW**	Return of an SRD response buffer after REPLY_WITHDRAW request
(0x12)	**L2_MAC_INDICATION_LOW_INVALID**	Return of so-called invalid indications (See MAC_INDICATION_LOW_INVALID service.) (only occurs with AMPRO2-SPC and AMPRO2-SPC-DP)

3.3 Interface Models

The interface between layer 2 and the layer-2 user is provided by a so-called application block interface. The application blocks which have already been mentioned are used for this interface.

3.3.1 Task Interface with Operating System Environment (L2_TASK_IFA_OS)

REQUESTs (i.e., jobs) are transferred from a layer-2 user to layer 2 by sending an application block to the mailbox of the layer-2 task via the operating system.

Available functions shown by example MTK-E111:

	Transfer a REQUEST ...
mtk_m_send (l2 task ID, application block pointer);-	from task level:
mtk_mi_send (l2 task ID, application block pointer);	from a certain besy-controlled interrupt handler: The related CONFIRMATION is always reported via the mailbox of a layer-2 user task
l2 task ID:	Task identifier of the layer-2 task = receiver task of the application block
Application block pointer:	Pointer to an L2 application block: The application block contains the job-related parameters of the REQUEST.

All **CONFIRMATIONs** (i.e., acknowledgments) and all **INDICATIONs** (i.e., events) of layer 2 are reported via the operating system by sending the related application block to the mailbox of a layer-2 user task. This layer-2 user task must be specified in the REQUEST under the "subsystem" parameter in the application block. The layer-2 user can fetch the CONFIRMATIONs and INDICATIONs from the mailbox via an operating system call.

Available function shown by MTK-E111:

mtk_m_receive (Address of an application block pointers);	Fetch a CONFIRMATION or INDICATION from the mailbox at task level
Application block pointer:	Pointer to an L2 application block: A received application block contains the job-related parameters of a CONFIRMATION or an INDICATION.

3.3.2 Calling Interface with Operating System Environment (L2_CALL_IFA_OS)

REQUESTs (i.e., jobs) are transferred from a layer-2 user to layer 2 by calling a layer-2 function.

Available functions:

l2_init ();	Initialization of the layer-2 interface: This function must be executed once before the FLC_FMA_MAC_RESET service.
return_value = l2_req (application block pointer); *return_value = l2_req_int (application block pointer);*	Transfer a REQUEST with "synchronous" processing ... from task level: from a certain besy-controlled interrupt handler: (See also notes on use of the layer-2 function "l2_req" or "l2_req_int" at the end of the section.)
return_value = L2_DIR_CON:	The service was immediately processed "synchronously" by layer 2 and is confirmed directly. The related CONFIRMATION is immediately available to the user in the same application block, and can be evaluated.
return_value = L2_NOT_DIR_CON:	Although the service was accepted by layer 2, it could not be processed immediately. The related CONFIRMATION is reported later via the mailbox of a layer-2 user task.
l2_req_ret_by_mbx (application block pointer); *l2_req_int_ret_by_mbx (application block pointer);*	Transfer of a REQUEST ... from task level: from a certain besy-controlled interrupt handler: The related CONFIRMATION is always reported via the mailbox of a layer-2 user task.

Application block pointer:	Pointer to an L2 application block: The application block contains the job-related parameters of the REQUEST.

All **CONFIRMATIONs** (i.e., acknowledgments) which are not processed "synchronously" by layer 2 and all **INDICATIONs** (i.e., events) are reported via the operating system by sending the related application block to the mailbox of a layer-2 user task. This layer-2 user task must be specified in the REQUEST under the "subsystem" parameter in the application block. The layer-2 user can fetch the CONFIRMATIONs and INDICATIONs from the mailbox via an operating system call.

Available function shown in example MTK-E111:

mtk_m_receive (address of an application block pointer);	Fetch a CONFIRMATION or an INDICATION from the mailbox at task level

Application block pointer:	Pointer to an L2 application block: A received application block contains the job-related parameters of a CONFIRMATION or an INDICATION.

3.3.3 Calling Interface without Operating System Environment (L2_CALL_IFA_CALLBACK)

REQUESTs (i.e., jobs) are transferred from the layer-2 user to layer 2 by calling a layer-2 function.
The "**subsystem**" parameter in the application block **never applies** to REQUESTs.

Available functions:

l2_init (lock_l2_req_fct_ptr, unlock_l2_req_fct_ptr); Initialization of the layer-2 interface:
This function must be executed once during software startup before the FLC_FMA_MAC_RESET service.

lock_l2_req_fct_ptr: Pointer to a user function which disables all processing levels with "l2_req ()" calls
unlock_l2_req_fct_ptr: Pointer to a user function which reenables all processing levels with "l2_req ()"call

return_value = l2_req (application block pointer); Transfer a REQUEST with "synchronous" processing (See also notes on use of layer-2 function "l2_req" or "l2_req_int"" at the end of this section.)

return_value = L2_DIR_CON: The service was immediately processed "synchronously" by layer 2 and is confirmed directly. The related CONFIRMATION is immediately available to the user in the same application block, and can be evaluated.

return_value = L2_NOT_DIR_CON: Although the service was accepted by layer 2, it could not be processed immediately. The related CONFIRMATION can be fetched later by the user with the "l2_con_ind" function.

Application block pointer: Pointer to an L2 application block:
The application block contains the job-related parameters of the REQUEST.

☞ The "l2_req ()" function must be protected against multiple calls at different levels (e.g., call from main level and from various interrupt levels).
This lock is provided internally by AMPRO2. Two function pointers to user functions are transferred for the "l2_init ()" function. These functions are called by AMPRO2 within the "l2_req ()" function.
The user must disable or reenable all levels with "l2_req ()" calls in accordance with the user environment.

☞ Adhere to the AMPRO2 application notes for the above locking mechanisms.

☞ The "l2_init ()" and "l2_con_ind ()" functions are not recursive and may only be used by the user at one level. Mutual calling of these functions must also be locked.

All **CONFIRMATIONs** (i.e., acknowledgments) which are not processed "synchronously" by layer 2 and all **INDICATIONs** (i.e., events) must be fetched by the layer-2 user by calling the layer-2 function "l2_con_ind" (polling).

Available function:

Application block pointer = l2_con_ind (); Polling of a CONFIRMATION or INDICATION:
When no CONFIRMATION or INDICATION is currently stored intermediately in layer 2, application block pointer = 0 is returned.
When a CONFIRMATION or an INDICATION is present in layer 2, it is returned via the application block pointer. High-priority CONFIRMATIONs and INDICATIONs are reported first.

Application block pointer: 0 or pointer to an L2 application block:
A received application block contains the job-related parameters of a CONFIRMATION or an INDICATION.

Notes on use of layer-2 function "l2_req" or "l2_req_int":

When the layer-2 function "l2_req" or "l2_req_int" is used for transferring REQUESTs to layer 2, services which can be "synchronously" processed by layer 2 are confirmed directly (return_value = L2_DIR_CON). The related CONFIRMATION is immediately available to the user in the same application block, and can be evaluated.
This mechanism distinguishes between services which are always confirmed directly and services which are only directly confirmed when errors occur.

- ☞ The "**subsystem**" parameter in the application block **never** applies to REQUESTs which are always confirmed directly.
- ☞ The services which the FLC can execute without the MAC are usually processed "synchronously."

The following are always confirmed directly (i.e., synchronously).

Jobs with an application block from an invalid memory area
(only occurs with AMPRO2-ASPC2 with *PTR_ATTR_FAR*)
(AMPRO2 acknowledges with *opcode* = *L2_FLC_CONFIRM_INVALID_APB*.)
Jobs with invalid *opcode* (AMPRO2 acknowledges with *opcode* = *L2_FLC_CONFIRM_INVALID_OPCODE*.)
FMA jobs with invalid *service_code* (AMPRO2 acknowledges with *opcode* = *L2_FMA_CONFIRM* and *L2_STATUS_IV*.)
MAC repeat auxiliary jobs with invalid *req_fc* (AMPRO2 acknowledges with *opcode* = *L2_REP_CONFIRM* and *L2_STATUS_IV*.)
FLC jobs with invalid *req_fc* (AMPRO2 acknowledges with *opcode* = *L2_FLC_CONFIRM* and *L2_STATUS_IV*.)
FLC repeat jobs with invalid *req_fc* (AMPRO2 acknowledges with *opcode* = *L2_FLC_CONFIRM* and *L2_STATUS_IV*.)

Layer-2 services which are always confirmed directly (i.e., synchronously)

FLC_FMA_MAC_RESET
MAC_RESET
TIME_TTH_READ
SAP_ACTIVATE
SAP_MODIFY
SAP_DEACTIVATE
SAP_STATUS_READ
LAS_READ
MAC_EVENT_RESOURCE_WITHDRAW
LAS_EVENT_RESOURCE_WITHDRAW
SAP_LOCK
SAP_UNLOCK
MAC_REQ_UNLOCK
MAC_REQ_WITHDRAW
CLEAR_MODE_ACTIVATE
CLEAR_MODE_DEACTIVATE
USER_TIMER_START
USER_TIMER_STOP
WATCHDOG_UPDATE
MAC_STATUS_READ

MAC_REPEAT_EXCHANGE_SYNCHRON
MAC_REPEAT_WITHDRAW

IND_RESOURCE_WITHDRAW
REPLY_UPDATE (bei AMPRO2-SPC und AMPRO2-SPC-DP)
REPLY_WITHDRAW

FLC_REPEAT_EXCHANGE
FLC_REPEAT_STATUS_READ

Layer-2 services which are only confirmed directly (i.e., synchronously) when the following errors occur

MAC_EVENT_RESOURCE_PROVIDE:	All states "L2_STATUS_NO" and all "L2_STATUS_IV"
LAS_EVENT_RESOURCE_PROVIDE:	All states "L2_STATUS_NO" and all "L2_STATUS_IV"
MAC_REQ_LOCK:	All states "L2_STATUS_NO" and all "L2_STATUS_IV"
SDA:	All resp states "L2_STATUS_IV"
SDN:	All resp states "L2_STATUS_IV"
SRD:	All resp states "L2_STATUS_IV"
FDL_STATUS:	All resp states "L2_STATUS_IV"
NOP:	All resp states "L2_STATUS_IV"
SDA-REPEAT:	All resp states "L2_STATUS_IV"
SDN-REPEAT:	All resp states "L2_STATUS_IV"
SRD-REPEAT:	All resp states "L2_STATUS_IV"
FDL_STATUS-REPEAT:	All resp states "L2_STATUS_IV"
NOP-REPEAT:	All resp states "L2_STATUS_IV"
MAC_REPEAT_EXCHANGE:	All resp states "L2_STATUS_NO" and all "L2_STATUS_IV"
COLLECT_SERVICE:	All resp states "L2_STATUS_IV"
DP_COLLECT_SERVICE:	All resp states "L2_STATUS_IV"
DP_FPT_COLLECT_SERVICE:	All resp states "L2_STATUS_IV"
IND_RESOURCE_PROVIDE:	All resp states "L2_STATUS_NO" and all "L2_STATUS_IV"
REPLY_UPDATE_SINGLE:	All resp states "L2_STATUS_NO" and all "L2_STATUS_IV"
REPLY_UPDATE_MULTIPLE:	All resp states "L2_STATUS_NO" and all "L2_STATUS_IV"
IND_RESOURCE_REPEAT_PROVIDE:	All resp states "L2_STATUS_NO" and all "L2_STATUS_IV"

3.4 Application Blocks

The interface between layer 2 and the layer-2 user is provided by a so-called application block interface. The application blocks which have already been mentioned are used for this purpose. All layer-2 application blocks are type "*struct l2_appl_blk*".

The transfer mechanisms have already been described in the preceding sections. Below is the physical setup of the application blocks.

L2 FMA application block:

Chain control pointer	*	*next_blk_ptr*
	*	*prev_blk_ptr*
Management data	UBYTE	*opcode*
	UBYTE	*subsystem*
	* or UWORD	*id_ptr*
FMA header	UBYTE	*as.fma.service_code*
	UBYTE	*as.fma.status*
	*	*as.fma.ptr1*
	*	*as.fma.ptr2*
	UBYTE	*as.fma.length*
	UBYTE	*as.fma.sap_nr*
	...	*as.fma. ...* (service-related)

L2 FLC/L2 MAC application block with layer-2 data buffers:

Chain control pointer	*	*next_blk_ptr*
	*	*prev_blk_ptr*
Management data	UBYTE	*opcode*
	UBYTE	*subsystem*
	* or UWORD	*id_ptr*
Response header	*	*as.flc.ptr1.resp_ptr*
	UBYTE	*as.flc.resp_length*
	UBYTE	*as.flc.resp_status*
Request header	*	*as.flc.ptr2.req_ptr*
	UBYTE	*as.flc.req_length*
	UBYTE	*as.flc.rem_adr*
	UBYTE	*as.flc.loc_adr*
	UBYTE	*as.flc.req_fc*
	UBYTE	*as.flc.rem_sap*
	UBYTE	*as.flc.loc_sap*
L4 header		(Optional)

L2 FLC/L2 MAC application block with chained application blocks:

Chain control pointer	*	*next_blk_ptr*
	*	*prev_blk_ptr*
Management data	UBYTE	*opcode*
	UBYTE	*subsystem*
	* or UWORD	*id_ptr*
Response header	*	*as.flc.ptr1.queue_ptr*
	UBYTE	*as.flc.resp_length*
	UBYTE	*as.flc.resp_status*
Request header	*	*as.flc.ptr2.queue_ptr*
	UBYTE	*as.flc.req_length*
	UBYTE	*as.flc.rem_adr*
	UBYTE	*as.flc.loc_adr*
	UBYTE	*as.flc.req_fc*
	UBYTE	*as.flc.rem_sap*
	UBYTE	*as.flc.loc_sap*

L2 FLC/L2 MAC application block for MAC repeat auxiliary services:

Chain control pointer	*	*next_blk_ptr*
	*	*prev_blk_ptr*
Management data	UBYTE	*opcode*
	UBYTE	*subsystem*
	* or UWORD	*id_ptr*
Response header	*	*as.flc.ptr1.apb_ptr*
	UBYTE	*as.flc.resp_length*
	UBYTE	*as.flc.resp_status*
Request header	*	*as.flc.ptr2.data_ptr*
	UBYTE	*as.flc.req_length*
	UBYTE	*as.flc.rem_adr*
	UBYTE	*as.flc.loc_adr*
	UBYTE	*as.flc.req_fc*
	UBYTE	*as.flc.rem_sap*
	UBYTE	*as.flc.loc_sap*

Short description of the parameters:

The **chain control pointers** are used to chain several application blocks.
The ***opcode*** parameter is used for precise specification of the service primitive. See section on service primitives.
The ***subsystem*** parameter contains the task identifier of the desired receiver task for CONFIRMATION or INDICATION for the interface models with operating system environments. This is usually the same task which initiated the corresponding REQUEST.
The ***id_ptr*** parameter is available to the user for use as desired. It remains unchanged when the application block is processed by layer 2.
The remaining layout of the application block depends on the service involved and is described in detail in the section on service-related interfaces.

3.5 Buffer Handling between Layer 2 and Layer-2 User

During dynamic operation of layer 2, memory blocks containing information and data are exchanged between layer 2 and the layer-2 user. The amount of memory required depends on the particular service involved.

Since AMPRO2 does not have its own memory management, the layer-2 user must provide layer 2 with all required memory blocks. After using them, layer 2 returns these memory blocks to the layer-2 user.

Basic rules:
The **application block** returned for a CONFIRMATION is the same physical block as the corresponding REQUEST.
Example: SAP_ACTIVATE CONFIRMATION and corresponding SAP_ACTIVATE REQUEST
The **application block** returned for an INDICATION is the same physical application block as the corresponding REQUEST.
Example 1: SDA INDICATION and corresponding IND_RESOURCE_PROVIDE REQUEST
Example 2: MAC_EVENT INDICATION and corresponding MAC_EVENT_RESOURCE_PROVIDE REQUEST

3.6 Integrated Memory Management

AMPRO2 offers the user integrated memory management. This memory management system can be used to dynamically manage application blocks of 2 different sizes (APBx with "PTR_ATTR" attribute), memory blocks of variable size (VAR with "PTR_ATTR" attribute) and data blocks of 4 different sizes (DBx with "L2_DATA_PTR_ATTR" attribute).
APBx memory management is recommended for the management of the AMPRO2 application blocks.
VAR memory management is recommended for the management of the layer-2 init block, layer-2 bus parameter block, SCB, LAS buffer, MAC-EVENT buffer, LAS-EVENT buffer and the SAP lock list.
DBx memory management is recommended for the management of the AMPRO2 telegram buffer.
The various types of memory have separate initialization functions. A large memory area is transferred to AMPRO2 with these initialization functions. The integrated memory management system divides this memory area into individual memory blocks of the required type and requested size. These individual memory blocks can then be requested dynamically by the AMPRO2 memory management system and released again.

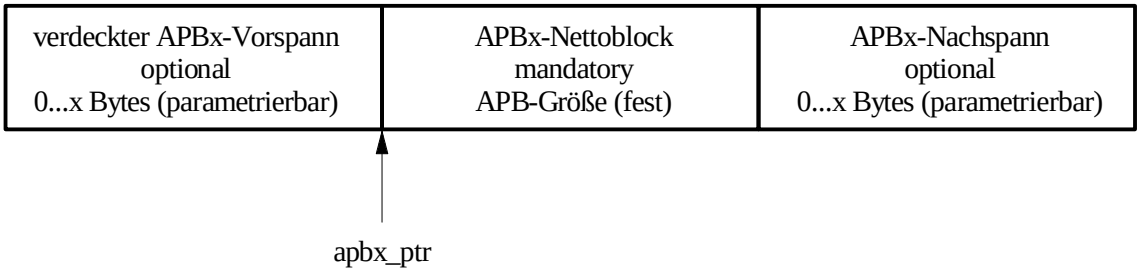
☞ All sizes are specified in bytes.

☞ In addition, otherwise allocated memory blocks can be used as AMPRO2 buffer, provided the special features of the particular AMPRO2 version are considered.

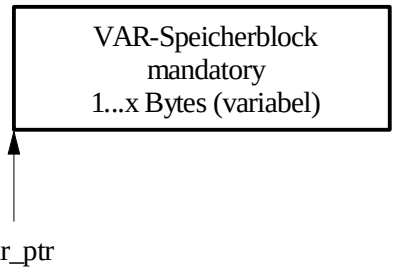
☞ The following additional requirements apply to the AMPRO2 ASPC2.
- The memory areas transferred with the initialization functions of the various AMPRO2 memory types must be located at even-numbered addresses.
- All sizes must be even numbers.
  - All leaders and trailers for the initialization functions
  - The net sizes of data blocks for "l2_mem_init_dbx"
  - The requested memory sizes for "l2_mem_alloc_var"
- If necessary, the user must provide filler bytes for all the above cases.
- If these requirements are not met, AMPRO2 returns a negative acknowledgment.

Layout of the various AMPRO2 memory types

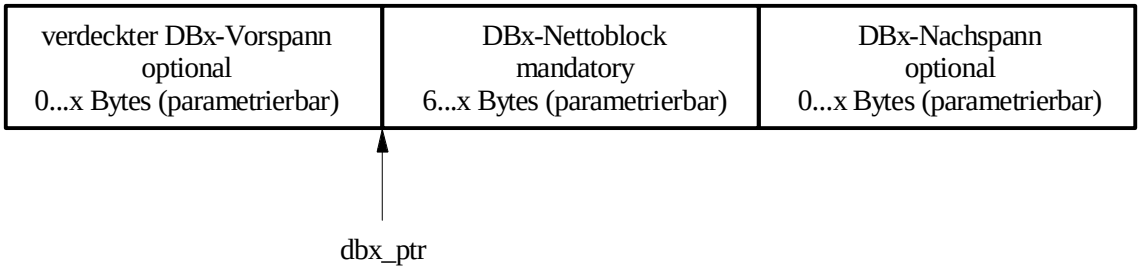
Speichertyp APBx:



Speichertyp VAR:



Speichertyp DBx:



Memory sizes for the various AMPRO2 memory types			
		Maximum permissible total length for one single memory block (can be parameterized) (theoretical size)	Maximum permissible length of the transferable memory area with the respective initialization function
APBx:	PTR_ATTR_NEAR with 80C16X	64 KB	64 KB
	PTR_ATTR_NEAR with 80X86/V25PLUS	64 KB	64 KB
	PTR_ATTR_BASED_SEG with 80X86/V25PLUS	64 KB	64 KB
	PTR_ATTR_FAR with 80C16X	16 KB	16 KB Optional: Unlimited [1)]
	PTR_ATTR_FAR with 80X86/V25PLUS	64 KB	64 KB
VAR:	PTR_ATTR_NEAR with 80C16X	63 KB	63 KB
	PTR_ATTR_NEAR with 80X86/V25PLUS	63 KB	63 KB
	PTR_ATTR_BASED_SEG with 80X86/V25PLUS	63 KB	63 KB
	PTR_ATTR_FAR with 80C16X	16 KB	16 KB
	PTR_ATTR_FAR with 80X86/V25PLUS	63 KB	63 KB
DBx:	L2_DATA_PTR_ATTR_NEAR with 80C16X	64 KB	64 KB
	L2_DATA_PTR_ATTR_NEAR with 80X86/V25PLUS	64 KB	64 KB
	L2_DATA_PTR_ATTR_BASED_SEG with 80X86/V25PLUS	64 KB	64 KB
	L2_DATA_PTR_ATTR_FAR with 80C16X	16 KB	16 KB Optional: Unlimited [2)]
	L2_DATA_PTR_ATTR_FAR with 80X86/V25PLUS: in FLAT model: Other:	64 KB 64 KB	4 GB 64 KB
	L2_DATA_PTR_ATTR_HUGE with 80C16X	64 KB	Unlimited
	L2_DATA_PTR_ATTR_HUGE with 80X86/V25PLUS	64 KB	Unlimited

1) In 80C16X environments and with PTR_ATTR_FAR, the optional compiler switch "L2_MEM_DISABLE_DATA_PAGE_LIMIT" (in config.h) can be used for the initialization functions of the APBx memory types to cancel the 16-KB upper memory limit. An unlimited memory area can then be transferred. In this case, the AMPRO2 memory management system ensures that a requested APBx does not exceed page limits. If necessary, AMPRO2 memory management inserts gaps in the memory for this purpose.

2) In 80C16X environments and with L2_DATA_PTR_ATTR_FAR the optional compiler switch "L2_MEM_DISABLE_DATA_PAGE_LIMIT" (in config.h) can be used for the initialization functions of the DBx memory types to cancel the 16-KB upper memory limit. An unlimited memory area can then be transferred. In this case, AMPRO2 memory management system ensures that a requested DBx does not exceed page limits. If necessary, AMPRO2 memory management inserts gaps in the memory for this purpose.

General functions available:

l2_mem_init (lock_l2_mem_fct_ptr, unlock_l2_mem_fct_ptr);

Basic initialization of AMPRO2 memory management
This function must be executed once during software startup before using the memory management system.

lock_l2_mem_fct_ptr: Pointer to user function which disables all processing levels with function calls of AMPRO2 memory management.

unlock_l2_mem_fct_ptr: Pointer to user function which reenables all processing levels with function calls of AMPRO2 memory management.

☞ All the following functions of AMPRO2 memory management must be protected again multiple calls from different levels (e.g., call from both the main level and various interrupt levels). The call of different functions must also be mutually locked.
This lock is provided internally by AMPRO2. For this purpose, two function pointers to user functions are transferred with the "I2_mem_init()" function. These functions are called by AMPRO2 within every function of AMPRO2 memory management (exception: "I2_mem_init ()"). In these functions, the user must disable or reenable all levels with function calls of AMPRO2 memory management in accordance with the user environment.

☞ The notes on AMPRO2 applications must also be adhered to for the above locking mechanisms.

☞ The "I2_mem_init ()" function is not recursive and may only be called by the user on one level.

Available functions for the APBx application blocks with x = 1 to 2

counter = l2_mem_init_apbx (pre_reserved, post_reserved, mem_start_ptr, mem_end_ptr);

Initialization function for memory type APBx:
This function can be used to reset and initialize AMPRO2 memory management for APBx application blocks. This function can be executed more than once during run time.

pre_reserved: Size in bytes of an optional, hidden APBx leader
For example, this leader can be used for mailbox management data for the MTK-E111 operating system.

post_reserved: Size in bytes of an optional APBx trailer
For example, this trailer can be used for layer-4 header data.

mem_start_ptr, mem_end_ptr: Pointer to the first or last element of the transferred memory area. A memory area for a maximum number of 65535 APBx may be transferred.
mem_start_ptr 0 is not permitted.

counter = 0: Although APBx memory management has been reset, it could not be initialized due to a user error.
- mem_start_ptr = 0
- (mem_start_ptr points to an odd-numbered address.)
- (The size of the leader or trailer is an odd number.)

counter = 1...65535: APBx memory management has been reset and initialized.
"counter" points of the number of allocated APBx.

apb_ptr = l2_mem_alloc_apbx ();

An APBx is allocated via APBx memory management. Such an APBx can be reenabled with the "l2_mem_free_apb" function.

apb_ptr = 0: No APBx available
apb_ptr != 0: Pointer to allocated APBx

return_value = l2_mem_free_apb (apb_ptr);

An APBx is enabled via APBx memory management.
Note: Run time optimization of the "l2_mem_free_apb" function is obtained by allocating the memory areas for APB1 to APB2 to ascending addresses. In addition, the run time of this function is shorter when APB1 is enabled than when APB2 is enabled.

apb_ptr: Pointer to an APBx to be enabled

return_value = TRUE: APBx was enabled.
return_value = FALSE: Memory block is unknown and could not be enabled.

Available functions for variable-size memory blocks

counter = l2_mem_init_var (mem_start_ptr, mem_end_ptr);

Initialization function for memory type VAR:
This function can be used to reset and initialize AMPRO2 memory management for memory blocks of variable size (VAR).
This function can be executed more than once during run time.

mem_start_ptr, mem_end_ptr: Pointer to the first or last element of the transferred memory area. The transferred memory area must be at least 2 bytes in size.

counter = 0: Although VAR memory management was reset, it could not be initialized due to a user error.
- The transferred memory area does not have the minimum size.
- (mem_start_ptr points to an odd-numbered address.)

counter = 1: VAR memory management was reset and initialized.

var_ptr = l2_mem_alloc_var (size);

A variable-size memory block is allocated via VAR memory management. Such a memory block cannot be enabled again.

size: Size in bytes of the requested memory block. Value range: ≥ 1

var_ptr = 0: No memory block with the requested size available
(or memory block with an odd-numbered length was requested)

var_ptr != 0: Pointer to allocated memory block

Available functions for DBx data blocks with x =1 to 4:

counter = l2_mem_init_dbx (netto_size, pre_reserved, post_reserved, mem_start_ptr, mem_end_ptr);

Initialization function for memory type DBx:
This function can be used to reset and initialize AMPRO2 memory management for DBx data blocks.
This function can be performed more than once during run time.

netto_size: Net size in bytes of a DBx data block. Value range: ≥ 6
pre_reserved: Size in bytes of an optional, hidden DBx leader
post_reserved: Size in bytes of an optional DBx trailer
For example, this trailer can be used for the PROFIBUS telegram trailer for AMPRO2-EMUL165.
mem_start_ptr, mem_end_ptr: Pointer to the first or last element of the transferred memory area. A memory area for a maximum number of 65535 DBx may be transferred. mem_start_ptr 0 is not permitted.

counter = 0: Although DBx memory management was reset, it could not be initialized due to a user error.
- The net size of the data block is too small.
- mem_start_ptr = 0
- (mem_start_ptr points to an odd-numbered address.)
- (The net size of the data block is an odd number.)
- (The size of the leader or trailer is an odd number.)

counter = 1...65535: DBx memory management was reset and initialized.
"counter" points to the number of allocated DBx.

dbx_ptr = l2_mem_alloc_dbx ();

A DBx is allocated via DBx memory management. Such a DBx can be reenabled with the "I2_mem_free_db" function.

dbx_ptr = 0: No DBx available
dbx_ptr != 0: Pointer to allocated DBx

return_value = l2_mem_free_db (db_ptr);

A DBx is enabled via DB memory management.
Note: A run time optimization of the "I2_mem_free_db" function can be obtained by allocating the memory areas for DB1 to DB 4 to ascending addresses. In addition, the run time of this function is shorter when DB1 is enabled than when DB2 is enabled, and so on.

db_ptr: Pointer to DBx to be enabled

return_value = TRUE: DBx was enabled.
return_value = FALSE: Memory block is unknown and could not be enabled.

3.7 Special Features of the AMPRO2 ASPC2

- ☞ The AMPRO2 ASPC2 is an implementation of PROFIBUS layer 2 with ASIC support by ASPC2. Currently in existence are **implementations** on SIEMENS controllers SAB 80C165 and SAB 80C167 and the 80x86 processor family (WINDOWS).
- ☞ When the AMPRO2 ASPC2 is used, the **ASPC2 error report** must also be considered.
  This report is available from the project leader at AUT E 132.
- ☞ The AMPRO2 ASPC2 supports the **ASIC releases** ASPC2-STEP-B and ASPC2-STEP-C.
  AMPRO2 adjusts automatically to the respective ASIC Step.
- ☞ The AMPRO2 ASPC2 only supports **interface model** "L2_CALL_IFA_CALLBACK"
  (can be set in config.h).
- ☞ When theAMPRO2 ASPC2 is used with ASPC2 Step B in connection with SIEMENS controllers SAB 80C165 and SAB 80C167, only the **write mode** with byte-high-enable (BHE) is supported.
  When ASPC2 Step C or later is used, write mode with write-low/write-high (WRL/WRH) is also supported. This mode must be set on the hardware, however.
  See also ASPC2 specifications. Careful with compatibility.
- ☞ The AMPRO2 ASPC2 only supports receipt of **SD3 telegrams**. Sending of SD3 telegrams is not supported.
- ☞ **Address extensions**: The AMPRO2 ASPC2 does not support segment extensions.
  For SAP extensions, the AMPRO2 ASPC2 supports only source and destination extension at the same time (i.e., both local and remote SAP equal default SAP or both don't equal default SAP).
- ☞ The AMPRO2 ASPC2 supports the **default SAP**.
- ☞ For the **baud rates** supported by the AMPRO2 ASPC2, see the baud rate table.
  See the FLC_FMA_MAC_RESET service.
- ☞ The **services** supported by the AMPRO2 ASPC2 are marked separately (●) in the section on service-related interfaces.
- ☞ The AMPRO2 ASPC2 uses the following **FIFO sizes** for the different ASIC releases.
  - For ASPC2 STEP B: 64 bytes
  - For ASPC2 STEP C and later: 128 bytes (fixed)
- ☞ The AMPRO2 ASPC2 offers a **user timer** for use as desired with parameterizable interval time.
  See FMA service USER_TIMER_START.
- ☞ For improved support of **PROFIBUS-DP**, the AMPRO2 ASPC2 offers the following extra functions.
  See also ASPC2 specifications.
  - DP mode. See below.
  - Repeat services. See sections 1.1.6.3. and 1.1.6.6.
  - Consistency support for sending. See below.
  - Consistency support for receiving (blocked mode). See below.
  - Consistency control logic support for initiator. See below.
  - Consistency control logic support for responder. See below. Not available with ASPC2 STEP B.

☞ **DP mode** can be parameterized on the AMPRO2 ASPC2 instead of the normal operating mode.
The parameter "*aspc2_par.l2_mode*" in the layer-2 init-block must be set to "*L2_PROFIBUS_DP*" for the FLC_FMA_MAC_RESET service.
In DP mode, the following changes apply to the AMPRO2 ASPC2. See also ASPC2 specifications.

- During token handling, Min_Slave_Intervall is monitored in accordance with PROFIBUS-DP for the active station instead of the setpoint token rotation time (TTR). After receipt of a token telegram, the AMPRO2 ASPC2 checks to determine whether Min_Slave_Intervall has expired for this station.
  If Min_Slave_Intervall has expired, the AMPRO2 ASPC2 locally starts Min_Slave_Intervall again and sends a GAP request. The normal GAP procedure is disabled for this. The MAC job list is then processed. See below.
  If Min_Slave_Intervall has not yet expired, the token is immediately passed to the next station.
- For active stations during the FLC_FMA_MAC_RESET service, the AMPRO2 ASPC2 automatically inserts a force-pass-token (FPT) with repeat function in the low-priority MAC job list. After processing by the MAC, this internal FPT is automatically put at the end of the low-priority MAC job list, and thus continuously specifies the cycle for token passing.
  This cycle consists of a GAP scan and all jobs from the MAC job list up to the concluding internal FPT.
  The internal FPT job is imaged in the AMPRO2 ASPC2 on a "NOP" service or on an SDN-LOW on the own station for ASPC2 STEP B.
- During this cycle, the user can insert the following jobs for active stations.
  All MAC repeat jobs (*opcode = L2_MAC_REP_REQUEST*) are immediately entered in the low-priority MAC job list. It is these jobs which primarily determine the cycle.
  All high-priority MAC jobs (*opcode = L2_MAC_REQUEST_HIGH*) are immediately entered in the high-priority MAC job list and can under some conditions increase the length of the cycle considerably (responsibility of the user).
  With low-priority MAC jobs (*opcode = L2_MAC_REQUEST_LOW*), a maximum of 1 job per cycle is entered in the low-priority MAC job list. Additional service requests for low-priority MAC jobs are stored intermediately in layer 2, and entered sequentially in the low-priority MAC job list during subsequent cycles.

☞ The AMPRO2 ASPC2 offers special **consistency support for sending**.
- For the initiator, the transfer of L2 sending data from external memory to the FIFO is performed with a lock cycle when sending request telegrams with a user data length less than or equal to 122 bytes (ASPC2 STEP B: 58 bytes).
- For the responder, the transfer of L2 sending data from external memory to the FIFO is performed with a lock cycle when sending response telegrams with a user data length less than or equal to 122 bytes (ASPC2 STEP B: 58 bytes).
- The sending procedure is not started until at least one L2 user data byte has been transferred to the FIFO which means that access authorization for consistent data existed.

  Exception: When the ASPC2 STEP B is used, the sending procedure is already started for the initiator after at least one L4 header data byte has been transferred to the FIFO.

This consistency support is always active during sending.

☞ The AMPRO2 ASPC2 also offers special **consistency support for receiving** - the so-called **blocked mode**. In this mode, the following procedures in the AMPRO2 ASPC2 are performed differently.
- For the initiator, the transfer of receiving data from the FIFO to external memory is not performed until a completely correct telegram receipt occurs when receiving response telegrams with a user data length less than or equal to 122 bytes (ASPC2 STEP B: 58 bytes).
- For the responder, the transfer of receiving data from the FIFO to external memory is not performed until a completely correct telegram receipt occurs when receiving request telegrams with a user data length less than or equal to 122 bytes (ASPC2 STEP B: 58 bytes).
- In both cases, the transfer of L2 receiving data is performed with a lock cycle.
- In non-blocked mode or when larger amounts of user data are involved, the transfer is performed immediately when the FIFO threshold is reached.

Blocked mode can be activated with the "*aspc2_par.blocked_mode*" parameter in the layer-2 init-block during the LC_FMA_MAC_RESET service.

Exception: On the ASPC2 STEP B, this mode cannot be explicitly activated. Blocked mode is automatically activated in DP mode. Otherwise it is always deactivated.

Caution: Blocked mode is mandatory when the ASPC2 transfers consistent L2 receiving data directly to the image memory. In addition, blocked mode is required when the user performs consistent direct accesses to the L2 receiving data without using the AMPRO2 MAC_REPEAT_EXCHANGE or FLC_REPEAT_EXCHANGE service.
In all other cases, blocked mode should not be used since it can slow down bus reaction time.

☞ The AMPRO2 ASPC2 also offers **consistency control logic support for the initiator** for both the request and response directions. See also ASPC2 specifications. The RDCONS signal can be activated for a MAC job for the duration of the transfer of the request data from the request buffer (sending buffer) to the ASPC2 FIFO. Similarly, the WRCONS signal can be activated during the transfer of the response data from the ASPC2 FIFO to the response buffer (receiving buffer). One or both consistency controls per MAC job can be used simultaneously.

In the application block, parameter "*as.flc.req_fc*" contains so-called consistency control bits for activation of consistency control logic support. By using or-logic (addition), these control bits can be set with the following Req-Fc masks, thus enabling the applicable consistency control.

Req-Fc mask for RDCONS: *L2_RDCONS* (0x10)
Req-Fc mask for WRCONS: *L2_WRCONS* (0x20)

Affected services: SDA-REQUEST, SDA-CONFIRMATION,
SDN-REQUEST, SDN-CONFIRMATION,
SRD-REQUEST, SRD-CONFIRMATION,
SDA-REPEAT-REQUEST, SDA-REPEAT-NEGATIVE-CONFIRMATION,
SDN-REPEAT-REQUEST, SDN-REPEAT-NEGATIVE-CONFIRMATION,
SRD-REPEAT-REQUEST, SRD-REPEAT-NEGATIVE-CONFIRMATION,
MAC_REPEAT_APB_WITHDRAW-CONFIRMATION

Prerequisite for use of consistency control logic support is, among others, the correct setting of "*aspc2_par.diagnose_port*" in the layer-2 init-block for the FLC_FMA_MAC_RESET service.

Exception: The AMPRO2 ASPC2 does not offer consistency control logic support for layer-4 header data.

☞ The AMPRO2-ASPC2 also offers **consistency control logic support for the responder** for both request and response directions. See also ASPC2 specifications. The WRCONS signal can be activated for receipt of a telegram during the transfer of the request data from the ASPC2 FIFO to the request buffer (receiving buffer). Similarly, the RDCONS signal can be activated during the transfer of the response data from the response buffer (sending buffer) to the ASPC2 FIFO. One or both consistency controls per telegram receipt can be used simultaneously.
Exception: The ASPC2 STEP B does not offer consistency control logic support for the responder.

In the application block, the "*as.flc.req_fc*" parameter contains so-called consistency control bits for activation of consistency control logic support. By using or-logic (addition), these control bits can be set with the following Req-Fc masks, thus activating the applicable consistency control. With the ASPC2 STEP B, these consistency control bits are irrelevant to requests and can thus be disregarded for AMPRO2.

Req-Fc mask for WRCONS: *L2_WRCONS* (0x20)

Affected services:
IND_RESOURCE_PROVIDE-REQUEST,
IND_RESOURCE_PROVIDE-CONFIRMATION,
IND_RESOURCE_REPEAT_PROVIDE-REQUEST,
IND_RESOURCE_REPEAT_PROVIDE-CONFIRMATION,
IND_RESOURCE_WITHDRAW-INDICATION,
SDA-INDICATION,
SDN-INDICATION,
SRD-INDICATION

Req-Fc mask for RDCONS: *L2_RDCONS* (0x10)

Affected services:
REPLY_UPDATE_SINGLE-REQUEST,
REPLY_UPDATE_SINGLE-CONFIRMATION,
REPLY_UPDATE_SINGLE-INDICATION,
REPLY_UPDATE_MULTIPLE-REQUEST,
REPLY_UPDATE_MULTIPLE-CONFIRMATION,
REPLY_UPDATE_MULTIPLE-INDICATION,
REPLY_WITHDRAW-INDICATION

Prerequisite for use of consistency control logic support is, among others, the correct setting of "*aspc2_par.diagnose_port*" in the layer-2 init-block during the FLC_FMA_MAC_RESET service.
Exception: The AMPRO2 ASPC2 does not offer consistency control logic support for layer-4 header data.

☞ The AMPRO2 ASPC2 also offers a special **clear mode** for the sending direction for an initiator. See also ASPC2 specifications. Among others, this mode can be used for the clear function for a DP master (DPM).
Prior to transfer to AMPRO2, all MAC repeat request jobs which are to be included in the clear function later are selected (i.e., marked) for clear mode.
Clear mode can be dynamically activated or deactivated later with the CLEAR_MODE_ACTIVATE or CLEAR_MODE_DEACTIVATE FMA services.
When clear mode is activated, all MAC repeat request jobs which are currently in AMPRO2 and which were selected for clear mode before transfer to AMPRO2 are included in the clear function.
The following MAC jobs can be included in the clear function.
SDA-REPEAT-REQUEST, SDN-REPEAT-REQUEST, and SRD-REPEAT-REQUEST
The clear function has two modes.

Clear-Mode-Data: In this mode, AMPRO2 sets the L2 data bytes to 00H when reading from the sending buffer (request buffer) for the selected MAC repeat request jobs. This causes zeros to be sent instead of the data from the sending buffer. The amount of sending data remains unchanged.

Clear-Mode-Length: In this mode, AMPRO2 sets the length of the L2 data bytes to 00H when reading from the sending buffer (request buffer) for the selected MAC repeat request jobs. In other words, the data from the sending buffer are not sent.

Caution: Clear mode has no effect on any layer-4 header data which may exist. These data are sent unchanged in clear mode.

Exception: The ASPC2 STEP B does not support clear mode.

All MAC repeat request jobs which are to be included in the clear function must be selected before transfer to AMPRO2. In the application block, the "*as.flc.loc_adr*" parameter contains so-called clear control bits. These control bits can be set with the following Loc-Adr masks, thus selecting the jobs for the clear function.
These clear control bits do not apply to requests on the ASPC2 STEP B and thus do not pertain to AMPRO2.
A maximum of one clear mode per MAC repeat job can be selected simultaneously.

Loc-Adr mask for Clear-Mode-Data: *L2_CLEAR_DATA* (0x10)
Loc-Adr mask for Clear-Mode-Length: *L2_CLEAR_LEN* (0x08)

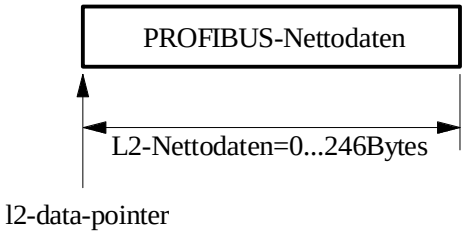
Affected services: SDA-REPEAT-REQUEST, SDA-REPEAT-NEGATIVE-CONFIRMATION, SDN-REPEAT-REQUEST, SDN-REPEAT-NEGATIVE-CONFIRMATION, SRD-REPEAT-REQUEST, SRD-REPEAT-NEGATIVE-CONFIRMATION, MAC_REPEAT_APB_WITHDRAW-CONFIRMATION

Caution: When clear mode is used (e.g., by the DPM), the clear control bits of all MAC repeat request jobs which are not to be included in the clear function must be set to a defined 0.
Careful when using new AMPRO2 applications which use clear mode!

Exception: The clear control bits do not apply to FDL_STATUS-REPEAT and NOP-REPEAT.

- The ASPC2 has an **address area** of 1 MB. The start address of this address area is transferred under "*aspc2_par.l2_mem_start_ptr_host*" in accordance with the hardware link in the layer-2 init-block.
- A so-called **system control block** (SCB) must be transferred to the AMPRO2 ASPC2 for the FLC_FMA_MAC_RESET service. This SCB is used for the internal coordination of FLC and ASPC2. The SCB and all **L2 application blocks** must be located in the AMPRO2 ASPC2 in a physically contiguous 64-K area. The start address of this 64-K memory area is transferred in the layer-2 init-block under "*aspc2_par.scb_apb_base_ptr_host*". The location of the SCB within this area is set with the "*aspc2_par.scb_offset*" parameter. The AMPRO2 ASPC2 supports the pointer attributes "near," "based," and "far" ("*PTR_ATTR*") for the SCB and the application blocks. An application block pointer of 0 is not permitted with the AMPRO2 ASPC2.
- On the AMPRO2 ASPC2, all **telegram buffers** are purely user data buffers (i.e., they contain only user data in accordance with PROFIBUS). Telegram buffers are all sending buffers (for request and response telegrams) and all receiving buffers (for request and response telegrams).
  The structure of all telegram buffers is shown below.

Telegramm-Puffer bei AMPRO2-ASPC2:



With the AMPRO2 ASPC2, the telegram buffers can be located within the entire memory area. The pointer attributes "near," "based", "far" and "huge" are supported ("*L2_DATA_PTR_ATTR*").
No telegram buffer is required for services which have a user data length of 0 bytes (e.g., SDA REQUEST with 0 bytes of user sending data).
Possible AMPRO2 ASPC2 accesses to the telegram buffers:

Sending buffer:	Read-accesses only
Receiving buffer:	Write-accesses also

- With the AMPRO2 ASPC2, the user must ensure that the base addresses of all L2 application blocks, all L2 telegram buffers and the system control block are located at even-numbered addresses (i.e., **word alignment**). The ASPC2 always accesses these memory areas by word.
  When the length of L2 telegram buffers is an odd number, the user must add a filler byte.
- As responder, the ASPC2 ASIC supports the default SAP and **L2 SAPs** in the parameterizable area from SAP 0 to a maximum of SAP 253. On the AMPRO2 ASPC2, this area is currently set to a constant SAP 0 to 63.
  The default SAP and SAPs 0 to 63 are supported as initiator.
- In contrast to PROFIBUS, **SAP 63** is not reserved on the AMPRO2 ASPC2 for the global access address. SAP 63 is handled the same as the other SAPs. Adherence to the PROFIBUS specifications for SAP 63 is left to the judgment of the layer-2 user.
  In accordance with PROFIBUS, SAP 63 is only permitted as the destination SAP for SDA and SDN jobs.
- The reaction of the AMPRO2 ASPC2 to the receipt of an unknown or **unsupported request telegram** is shown in the table on the plausibility checks during receipt of request telegrams (MAC indication). See the SAP_ACTIVATE service.
- The AMPRO2 ASPC2 requires receiving memory (indication resource/repeat indication resource) for the receipt of a REQUEST telegram (**MAC INDICATION**). Such a resource consists of application block and indication buffer. It must be provided by the user with the "IND_RESOURCE_PROVIDE/IND_RESOURCE_REPEAT_PROVIDE" service. The resources are transferred to layer 2 as related to SAP and are also managed by it in this way. Receipt of a REQUEST telegram for a SAP which has no resource is not processed further by layer 2 and is acknowledged negatively on the bus. The telegram is received correctly when a resource is available for this SAP.
  The AMPRO2 ASPC2 uses the SAP-related resources directly for telegram receipt (i.e., no alternating buffer system). After the telegram has been received correctly, the telegram is reported first in the SAP via MAC INDICATION when a "normal" indication resource is involved. When a repeat indication resource is involved, a repeat check is performed instead. See section 1.1.6.6.
  On the AMPRO2 ASPC2, the assignment of application block to indication buffer which existed with the "IND_RESOURCE_PROVIDE" service is retained for MAC INDICATIONs. When repeat indication resources are involved, the assignment of application block to indication buffer which existed with the "IND_RESOURCE_REPEAT_PROVIDE" service is only retained for MAC INDICATIONs when the "FLC_REPEAT_EXCHANGE" service is not executed. In addition, a SAP-related optimization of the length of the indication buffer can be performed.
- No MAC indication is triggered on the AMPRO2 ASPC2 for the responder for SDA and SRD services for which user data are not transferred in either the request or the response telegram (i.e., **empty polling**). In this case, the indication resource remains in layer 2.
  Exception: When SAP contains a repeat indication resource, empty polling is not checked and a repeat check is performed instead. See section 1.1.6.6.
  When empty polling is involved, the table on plausibility of receipt of a request telegram (MAC indication) must be adhered to. See the SAP_ACTIVATE service.
- On the AMPRO2 ASPC2, the correct **time sequence** of the reporting of MAC confirmations and MAC indications of the same priority is ensured. MAC confirmations and MAC indications are reported in the order in which they occurred.
  A different mechanism is used for reporting MAC repeat confirmations. This mechanism meets the requirements of PROFIBUS-DP.
- On the AMPRO2 ASPC2, layer 2 **SRD response data** of the responder can only be made available with the REPLY_UPDATE_SINGLE or REPLY_UPDATE_MULTIPLE service.

☞ The AMPRO2 ASPC2 supports the **layer-4 header** for request telegrams in both sending and receiving directions. Typical of this function is the fact that other data can be inserted at the beginning of the telegram user data portion before the data from the request buffer (e.g., layer-4 and/or layer-7 header data). One fixed header length and two parameterizable header lengths are supported. The parameterizable header lengths are set with the FLC_FMA_MAC_RESET service in the layer-2 init-block under "*aspc2_par.l4_hlen_var1*" and "*aspc2_par.l4_hlen_var2*".
The "*opcode*" parameter in the application block contains so-called L4 header control bits for activation of layer-4 header support. By using or-logic (addition), these control bits can be set with the following opcode masks, thus enabling the desired layer-4 header support.

Opcode mask for 2-byte fixed length:	*L2_L4_HLEN_FIXED*	(0x20)
Opcode mask for parameterizable length 1:	*L2_L4_HLEN_VAR1*	(0x40)
Opcode mask for parameterizable length 2:	*L2_L4_HLEN_VAR2*	(0x60)

Affected services: SDA-REQUEST, SDA-CONFIRMATION, SDA-INDICATION,
SDN-REQUEST, SDN-CONFIRMATION, SDN-INDICATION,
SRD-REQUEST, SRD-CONFIRMATION, SRD-INDICATION,
SDA-REPEAT-REQUEST, SDA-REPEAT-NEGATIVE-CONFIRMATION,
SDN-REPEAT-REQUEST, SDN-REPEAT-NEGATIVE-CONFIRMATION,
SRD-REPEAT-REQUEST, SRD-REPEAT-NEGATIVE-CONFIRMATION,
MAC_REPEAT_APB_WITHDRAW-CONFIRMATION,
IND_RESOURCE_PROVIDE-REQUEST, IND_RESOURCE_PROVIDE-CONFIRMATION,
IND_RESOURCE_WITHDRAW-INDICATION,
IND_RESOURCE_REPEAT_PROVIDE-REQUEST,
IND_RESOURCE_REPEAT_PROVIDE-CONFIRMATION

When layer-4 header support is used, the layer-4 header data are placed in the application block directly behind the request header. This additional storage requirement must be taken into consideration for the affected application blocks. With AMPRO2 memory management, this can be implemented with the APBx trailer. The "*as.flc.req_length*" parameter in the application block only specifies the amount of user data in the request buffer, thus providing the L2 telegram with user data ("*as.flc.req_length*" + layer-4 header length).

When the AMPRO2 ASPC2 as responder receives a request telegram whose user data length is less than the layer-4 header length selected in the corresponding indication resource, the AMPRO2 ASPC2 provides a negative acknowledgment to the bus. See the table on the plausibility of receipt of a request telegram (MAC indication) for the SAP_ACTIVATE service. In this case, the user does not receive a MAC indication. The indication resource remains in layer 2.

☞ The AMPRO2 ASPC2 performs a **plausibility check of the sending length** for MAC request jobs. If the addition of the user data length in the sending buffer (*req_length*) and any layer-4 header length which may be present results in an invalid (i.e., too large) value in accordance with PROFIBUS, this job is not sent to the bus and is negatively acknowledged with the status "*L2_STATUS_IL*". In addition, the MAC EVENT "*L2_REQ_LENGTH_ERROR*" is generated.
Exception: When the ASPC2 STEP B is used, AMPRO2 sends this "incorrect" job with a user data length of 0 without the layer-4 header data. However, the user receives a positive MAC confirmation. In addition, the MAC EVENT "*L2_REQ_LENGTH_ERROR*" is generated.

☞ A so-called alternating buffer system is used for **MAC EVENT resources**. The first buffer of this system must be transferred with the FLC_FMA_MAC_RESET service in the layer-2 init-block under "*first_mac_event_buf_ptr*". MAC EVENT buffers are available in various lengths.

☞ The AMPRO2 ASPC2 reveals to the user the **function for processing all ASPC2 interrupts** (i.e., provides flexibility). This function must be performed by the user when an ASPC2 interrupt occurs. The user can adapt the corresponding interrupt framework to his specific environment.

Syntax: *UWORD L2_IFA_CODE_ATTR l2_aspc2_int_handler (void);*

The return parameter contains the current value of ASPC2 register IMR (i.e., interrupt mask register).

3.8 Description of Service-Related Interfaces

This description of the service-related interfaces explains each layer-2 service as it relates to the service primitive (i.e., REQUEST, CONFIRMATION and INDICATION. The entire application block with relevant transfer parameters or the return data is described for each service primitive. This service-related description applies unchanged to all interface models contained in section 1.1.2. Special attention must be paid to the special features of the various AMPRO2 implementations. See section 1.1.5.
All C designators are shown in *italics*. All structures and constants required by the layer-2 user are included in the header file "l2_user.h".
A "P" following a parameter stands for its plausibility via AMPRO2. A "(P)" following a parameter stands for partial plausibility.

- ☞ When a job with an application block from an invalid memory area was transferred to layer 2, the AMPRO2 acknowledges with *opcode* = *L2_FLC_CONFIRM_INVALID_APB*.
  (Only occurs with the AMPRO2 ASPC2 with *PTR_ATTR_FAR*)
- ☞ When a job with an invalid or unsupported *opcode* was transferred to layer 2, AMPRO2 acknowledges with *opcode* = *L2_FLC_CONFIRM_INVALID_OPCODE*.
- ☞ When an FMA job with an invalid or unsupported *service_code* was transferred to layer 2, AMPRO2 acknowledges with *opcode* = *L2_FMA_CONFIRM* and *status* = *L2_STATUS_IV*.
- ☞ When a MAC repeat auxiliary job with an invalid or unsupported *req_fc* was transferred to layer 2, the AMPRO2 acknowledges with *opcode* = *L2_REP_CONFIRM* and *resp_status* = *L2_STATUS_IV*.
- ☞ When an FLC job with an invalid or unsupported *req_fc* was transferred to layer 2, AMPRO2 acknowledges with *opcode* = *L2_FLC_CONFIRM* and *resp_status* = *L2_STATUS_IV*.

3.8.1 FMA Services

3.8.1.1 FLC_FMA_MAC_RESET

FLC_FMA_MAC_RESET REQUEST

This service resets layer 2. In addition, this service transfers new initialization and bus parameters. All internal layer-2 data structures are initialized again. All jobs and user memory in layer 2 are lost.

- ☞ The FLC_FMA_MAC_RESET service must be executed as the first service during layer-2 startup.
- ☞ On the layer-2 task interface (L2_TASK_IFA_OS), additional jobs in the mailbox of the layer-2 task are not deleted during the FLC_FMA_MAC_RESET service (i.e., they are processed afterwards).

next_blk_ptr		Disregard
prev_blk_ptr		Disregard
opcode	P	*L2_FMA_REQUEST*
subsystem		Task identifier of the receiver task for the confirmation
id_ptr		Can be used as desired by the user
as.fma.service_code	P	*L2_FLC_FMA_MAC_RESET*
as.fma.status		Disregard
as.fma.ptr1		Pointer to completed layer-2 init-block
as.fma.ptr2		Pointer to completed layer-2 bus parameter block
as.fma.length		Disregard
as.fma.sap_nr		Disregard
as.fma. ...		Disregard

Layer-2 init-block for AMPRO2 ASPC2: ("*struct l2_init_blk*")

☞ See also ASPC2 specifications.

☞ Possible access: "*as.fma.ptr1 -> init_blk.first_mac_event_buf_ptr*"

first_mac_event_buf_ptr Pointer to first MAC EVENT buffer. Minimum length: *first_mac_event_buf_length* in layer-2 init-block
Remains in layer 2 for *L2_STATUS_OK*. First buffer for the alternating buffer system.
See also MAC_EVENT service.

first_mac_event_buf_length Length of the first MAC EVENT buffer: 1 to 255 bytes

user_error_fct_ptr Pointer to user error function. This function is called by layer 2 when serious errors are detected.
Function type: *void (*user_error_fct_ptr) (UBYTE error_code, void PTR_ATTR *error_ptr)*.
Layer 2 transfers an error code and an error pointer to the function. See also section 1.1.6.8.
Provision of error handling in this function is recommended so that layer 2 is returned to its correct status. A return to the calling program may not be made at the end of the user error function.

reset_asic_fct_ptr Pointer to user function which performs a hardware reset of the ASPC2. The hardware must be equipped with the RESET pin so that the ASPC2 can reset the hardware. Keep ASPC2 specifications in mind.
Function type: *void (*reset_asic_fct_ptr) (void);*

mask_int_asic_fct_ptr Pointer to user function which masks the ASPC2 interrupt
Function type: *void (*mask_int_asic_fct_ptr) (void);*
Note: All interrupts including the priority level of the ASPC2 interrupt are usually disabled within this function, or the ASPC2 interrupt is explicitly disabled. When explicit disabling is used, **any** other interrupt (also interrupts with lower priority) can extend this interrupt disable. When an interrupt disable is extended, operation of the ASPC2 and thus data flow may deteriorate due to AMPRO2.

unmask_int_asic_fct_ptr Pointer to user function which enables the ASPC2 interrupt again
Function type: *void (*unmask_int_asic_fct_ptr) (void);*

mask_int_global_fct_ptr Pointer to user function which globally masks all interrupts
Function type: *void (*mask_int_global_fct_ptr) (void);*
Note: This interrupt disable is required in addition to the "mask_int_asic_fct_ptr" disable for brief ASPC2 lock times within AMPRO2. If this disable is extended by other causes, excessively long ASPC2 lock times may increase bus reaction times or cause bus collisions. For this reason, a global interrupt disable is recommended.
When explicit disabling is used, **any** other interrupt (also interrupts with lower priority) can extend this interrupt disable. When a disable caused by increasing the priority level to a certain value occurs, any higher-priority interrupt can extend this interrupt disable.
Caution: Since the "mask_int_global_fct_ptr" interrupt disable is executed within the interrupt disable with "mask_int_asic_fct_ptr", an interrupt enable via "unmask_int_global_fct_ptr" may not cancel an interrupt disable by "mask_int_asic_fct_ptr".

unmask_int_global_fct_ptr Pointer to user function which globally enables all interrupts again.
Function type: *void (*unmask_int_global_fct_ptr) (void);*

aspc2_par.asic_adr. ...
as_mem_ptr
as_io_offset Start address of the 64-byte internal ASPC2 register record from the viewpoint of the host processor (in accordance with hardware integration)
- As pointer for allocation in the memory area
- As I/O offset for allocation in the I/O area. This I/O offset only applies when several AMPRO2s are used.

aspc2_par.l2_mem_start_ptr_host Start address of the 1-MB ASPC2 address area for communication memory from the viewpoint of the host processor
Corresponds to address 0 from the viewpoint of the ASPC2 (in accordance with hardware integration)
Value range:
80x86, V25: L2_DATA_PTR_ATTR_FAR:
PC_WIN_95, RMOS_FLAT: ptr = x
else: base = x, offset = 0
else: base = 0, offset = 0
8016x: irrelevant (Start address can be set in config.h via L2_MEM_START_PAGE_HOST)

aspc2_par.scb_apb_base_ptr_host Start address of the 64-KB memory area for the ASPC2 system control block (SCB) and L2 application blocks from the viewpoint of the host processor. Must be located in the 1-MB ASPC2 address area (in accordance with hardware integration).
Value range: PC_WIN_31, RMOS: *aspc2_par.l2_mem_start_ptr_host* in layer-2 init-block
PC_WIN_95, RMOS_FLAT:*aspc2_par.l2_mem_start_ptr_host* + offset,
whereby offset < 0xFFFFF (1 MB)
Caution: No own logical address may be obtained from the operating system. Basis is the logical address of the 1-MB ASPC2 address area.
else: base = x, offset = 0

...

aspc2_par.scb_offset P Offset of the ASPC2 system control block (SCB) within the 64-KB memory area. See above.
Value range:
ASPC2_ADR_MODE_LINEAR: 0
else: 80x86, V25: 0...(10000h - L2_LEN_SCB)
8016x: PTR_ATTR_NEAR: 0...(F000h - L2_LEN_SCB)
L2_APB_ATTR_FAR_16KB: 0...(4000h - L2_LEN_SCB)
L2_APB_ATTR_FAR_64KB: 0...(10000h - L2_LEN_SCB)
Note: In addition, the SCB must be located at an even-numbered address (i.e., word alignment) and have a minimum length of *L2_LEN_SCB* (l2_user.h).

aspc2_par.diagnose_port ASPC2 diagnosis mode for the diagnostic port:
L2_ASPC2_DIAG_MS_CONS: Partial diagnosis for micro-sequencer and consistency control signals WRCONS and RDCONS
L2_ASPC2_DIAG_MS: Diagnosis for micro-sequencer
L2_ASPC2_DIAG_KS_LOCK_CONS: Partial diagnosis for channel sequencer, BUSLOCKOUT and the WRCONS and RDCONS control signals
L2_ASPC2_DIAG_KS_LOCK: Diagnosis for channel sequencer and BUSLOCKOUT

aspc2_par.shared_dualp_mem Type of external communication memory:
L2_ASPC2_SHARED_MEM: Shared memory mode
L2_ASPC2_DUALP_MEM: Dual-port memory mode

aspc2_par.xreqrdy_delay Delay time between 'XENBUF active' and 'XREQRDY active ' in dual-port memory mode:
Value range for *L2_ASPC2_DUALP_MEM*: *L2_ASPC2_XREQRDY_DELAY_1_CLOCK*, *L2_ASPC2_XREQRDY_DELAY_3_CLOCKS*
Not applicable to *L2_ASPC2_SHARED_MEM*

aspc2_par.l2_mode Layer-2 param: *L2_PROFIBUS*: ASPC2 in normal operating mode
L2_PROFIBUS_DP: ASPC2 in DP mode. See chap. 1.1.5.1.
L2_PROFIBUS_MONITOR: ASPC2 in monitor mode
Only the FLC_FMA_MAC_RESET service is supported in monitor mode.

aspc2_par.holda_pol Polarity of "HOLDA" signals: *L2_ASPC2_HOLDA_POL_LOW, L2_ASPC2_HOLDA_POL_HIGH*

aspc2_par.int_pol Polarity of interrupt outputs: *L2_ASPC2_INT_POL_LOW, L2_ASPC2_INT_POL_HIGH*

aspc2_par.int_config ASPC2 interrupt mode: *L2_ASPC2_NOT_SEP_INT*: All interrupts on pin "INT-EV"
(*L2_ASPC2_SEP_INT* not supported by AMPRO2 ASPC2)

aspc2_par.start_bit_control ASPC2 start bit check: *L2_ASPC2_START_BIT_CONTROL_OFF*, *L2_ASPC2_START_BIT_CONTROL_ON*

aspc2_par.stop_bit_control ASPC2 stop bit check: *L2_ASPC2_STOP_BIT_CONTROL_OFF*, *L2_ASPC2_STOP_BIT_CONTROL_ON*

aspc2_par.tok_err_limit Threshold value for the number of non-plausible token telegrams per 256 consecutive token rotations
When reached, the MAC generates MAC EVENT "*L2_MAC_RESET_LAS_USELESS*" for active stations and goes to "listen token."
Value range: 1 to 255

aspc2_par.resp_err_limit Threshold value for the number of incorrect response telegrams per 16 consecutive request telegrams (exception: SDN)
When reached, the MAC generates MAC-EVENT "*L2_DOUBLE_TOKEN*" for active stations and goes to "active idle."
Value range: 1 to 15

aspc2_par.l4_hlen_var1 P Layer-4 header length in words: Value range: 1 to 16
(corresponding *L2_L4_HLEN_VAR1* opcode mask)

aspc2_par.l4_hlen_var2 P Layer-4 header length in words: Value range: 1 to 16
(corresponding *L2_L4_HLEN_VAR2* opcode mask)

aspc2_par.seg_0_wait_cnf Configuration of the external communication memory for ASPC2 accesses
Memory area: 0 to 256 KB:
General wait states: *L2_ASPC2_WAIT_1*... *L2_ASPC2_WAIT_4*
Ready activation: *L2_ASPC2_USE_EXT_RDY*, *L2_ASPC2_NOT_USE_EXT_RDY*
Additional wait state: *L2_ASPC2_USE_RDY_WAIT*, *L2_ASPC2_NOT_RDY_WAIT*
Note: The general wait states are inserted before Ready when Ready is activated. The additional settable wait state is inserted after Ready when Ready is activated. No additional wait state can be parameterized for *L2_ASPC2_WAIT_1*.
Configuration is performed by adding the setting of all three classes.
Example: *aspc2_par.seg_0_wait_cnf = L2_ASPC2_WAIT_2 + L2_ASPC2_USE_EXT_RDY + L2_ASPC2_NOT_RDY_WAIT*;

aspc2_par.seg_1_wait_cnf Configuration of the external communication memory for ASPC2 accesses
Memory area: 256 to 512 KB: ditto

aspc2_par.seg_2_wait_cnf Configuration of the external communication memory for ASPC2 accesses
Memory area: 512 to 768 KB: ditto

aspc2_par.seg_3_wait_cnf Configuration of the external communication memory for ASPC2 accesses
Memory area: 768 KB to 1 MB: ditto

...

aspc2_par.quick_access_mode
Optional quick-access mode for optimized data transfer between ASPC2 and external memory
Not applicable to ASPC2-STEP B since not supported
Value range for all other ASPC2-STEPs:
L2_ASPC2_QUICK_ACCESS_MODE_OFF, *L2_ASPC2_QUICK_ACCESS_MODE_ON*

aspc2_par.blocked_mode
Optional blocked mode for transfer of receiving data from ASPC2 to external memory
See also section 1.1.5.1.
Not applicable to ASPC2-STEP B: Blocked mode is automatically activated in DP mode.
Value range for all other ASPC2-STEPs:
L2_ASPC2_BLOCKED_MODE_OFF, *L2_ASPC2_BLOCKED_MODE_ON*
Caution: Blocked mode is mandatory when the ASPC2 transfers L2 user data directly to image memory. In addition, blocked mode is required when the user directly accesses the L2 user data buffer without using the MAC_REPEAT_EXCHANGE or FLC_REPEAT_EXCHANGE AMPRO2 services. In all other cases, blocked mode should not be used since it can increase bus reaction times.

aspc2_par.int_delay_time
Settable minimum time between two ASPC2 interrupts. Time starts with acknowledgment of EOI.
Not applicable to ASPC2-STEP B: 1µsec is always used here.
Value range for all other ASPC2-STEPs:
L2_ASPC2_INT_DELAY_1_US, *L2_ASPC2_INT_DELAY_1_MS*

aspc2_par.user_timer_base
Time interval for the internal AMPRO2 user timer (cyclic timer)
See also USER_TIMER_START FMA service.
Not applicable to monitor mode: Is not supported
Not applicable to ASPC2-STEP B: 2.1 sec is always used here.
Value range for all other cases:
L2_ASPC2_USER_TIMER_10_MS: Interval time of 10 msec
L2_ASPC2_USER_TIMER_2100_MS: Interval time of 2.1 sec

aspc2_par.user_timer_fct_ptr
Pointer to user function which is called cyclically each time the user timer expires.
The user timer must also be started.
See also USER_TIMER_START FMA service.
Function type: *void (*aspc2_par.user_timer_fct_ptr) (void);*
Not applicable to monitor mode: Is not supported
This parameter is also irrelevant when the user timer is never started.
Note: This user function is called each time the interval time expires for the ASPC2 interrupt handler. Since the user timer is scanned at the end of the ASPC2 interrupt handler, this user function can be delayed when many interrupts occur. The user timer can be dynamically started (USER_TIMER_START FMA service) and stopped (USER_TIMER_STOP FMA service).

monitor_par.filter_sd4
Filter for token telegrams
L2_ASPC2_SD4_FILTER_ON: Filter
L2_ASPC2_SD4_FILTER_OFF: No filter

monitor_par.filter_fdl_status
Filter for FDL status telegrams
L2_ASPC2_FDL_STATUS_FILTER_ON: Filter
L2_ASPC2_FDL_STATUS_FILTER_OFF: No filter

monitor_par.adr_selector_1
When an address selector is used, all telegrams to and from this station are recorded, but Broadcast telegrams are suppressed. When both selectors are used, only the telegrams between 2 defined stations are recorded.
Value range: *L2_ASPC2_NO_MON_ADR_SELECTOR*: Selector off
0 to 126: Selector on

monitor_par.adr_selector_2
Same as above
Caution: Selector 2 may only be activated when selector 1 is activated.

monitor_par.write_blk_ptr
Write pointer to the monitor list (i.e., chaining of the monitor application blocks)
Caution: Entry must be in ASIC format.

monitor_par.read_blk_ptr
Read pointer to the monitor list (i.e., chaining of the monitor application blocks)
Caution: Entry must be in ASIC format.

monitor_par.trigger_1_offset
Trigger offset in bytes of word 1 to be compared (in reference to the base address of the monitor application block)
Caution: The word to be compared must be located at an even-numbered address.

monitor_par.trigger_1_mask
Certain bits are selected or masked in word 1 to be compared (e.g., lower or higher byte).

monitor_par.trigger_1_compare
Comparison value 1
Trigger 1 is triggered when
word (trigger_1_offset) AND trigger_1_mask == trigger_1_compare

monitor_par.trigger_2_offset
Trigger offset in bytes of word 2 to be compared (in reference to base address of the monitor application block)
Caution: The word to be compared must be located at an even-numbered address.

monitor_par.trigger_2_mask
Certain bits are selected or masked in word 2 to be compared (e.g., lower or higher byte).

monitor_par.trigger_2_compare
Comparison value 2
Trigger 2 is triggered when
word (trigger_2_offset) AND trigger_2_mask == trigger_2_compare

...

monitor_par.timeout_fct_ptr
Pointer to user function which is called when a TIMEOUT event occurs
This is used to determine whether there is still any activity on the bus. Timeout for passive stations is greater than that of active stations.
Function type: *void (*monitor_par.timeout_fct_ptr) (void);*
Note: This user function is called within the ASPC2 interrupt handler when a TIMEOUT event occurs.

monitor_par.monitor_full_fct_ptr
Pointer to user function which is called when the monitor list is full.
Function type: *void (*monitor_par.monitor_full_fct_ptr) (void);*
Note: This user function is called within the ASPC2 interrupt handler when the monitor list is full.

monitor_par.monitor_trigger_fct_ptr
Pointer to user function which is called when a trigger procedure occurred
Function type: *void (*monitor_par.monitor_trigger_fct_ptr) (void);*
Note: This user function is called within the ASPC2 interrupt handler when a trigger procedure occurs.

windows_par.selector_l2_mem_seg_0
Additional parameter for support of the DP ISA card under Windows 3.1
Selector for RAM segment 0 of DP ISA card (memory area: 0 to 64 KB)

windows_par.selector_l2_mem_seg_1
Additional parameter for support of the DP ISA card under Windows 3.1
Selector for RAM segment 1 of DP ISA card (memory area: 64 to 128 KB)

windows_par.selector_l2_mem_seg_2
Additional parameter for support of the DP ISA card under Windows 3.1
Selector for RAM segment 2 of DP ISA card (memory area: 128 to 192 KB)

windows_par.selector_l2_mem_seg_3
Additional parameter for support of the DP ISA card under Windows 3.1
Selector for RAM segment 3 of DP ISA card (memory area: 192 to 256 KB)

Layer-2 init-block for AMPRO2 SPC2: ("*struct l2_init_blk*")

☞ The SPC2 is equipped with an internal 2-KB RAM. This RAM contains all telegram buffers for handling telegram communication on the bus. The organization of this RAM can be parameterized in the layer-2 init-block. See also notes on configuration of the internal SPC2 RAM at the end of the layer-2 init-block.

☞ See also SPC2 specifications.

☞ Possible access: "*as.fma.ptr1 -> init_blk.first_mac_event_buf_ptr*"

first_mac_event_buf_ptr		Pointer to first MAC EVENT buffer. Minimum length: *first_mac_event_buf_length* in layer-2 init-block Remains *L2_STATUS_OK* in layer 2. First buffer for the alternating buffer system. See also the MAC_EVENT service.
first_mac_event_buf_length	P	Length of the first MAC EVENT buffer: 1 to 255 bytes
ind_buf_length_min	P	Minimum user data length for all indication buffers of this station. See section 1.1.5. Maximum permissible user data length for all indication telegrams on this station. Minimum value: Minimum from (246, ((*spc2_par.ind_size* * 8) - 8)) bytes
user_error_fct_ptr		Pointer to user error function. This function is called by layer 2 when serious errors are detected. Function type: *void (*user_error_fct_ptr) (UBYTE error_code, void PTR_ATTR *error_ptr);* Layer 2 transfers an error code and an error pointer to the function. See also section 1.1.6.8 on error ouputs for AMPRO2. Error handling is recommended in this function so that layer 2 is returned to its correct status. A return to the calling program may not be made at the end of the user error function.
reset_asic_fct_ptr		Pointer to user function which performs a hardware reset of the SPC2. The hardware must be equipped with the RESET pin so that the SPC2 can be reset. Adhere to SPC2 specifications. Function type: *void (*reset_asic_fct_ptr) (void);*
mask_int_asic_fct_ptr		Pointer to user function which masks the SPC2 interrupt and also the collect interrupt for AMPRO2 operating system integration. Function type: *void (*mask_int_asic_fct_ptr) (void);* Note: All interrupts including the priority level of the SPC2 interrupt (collect interrupts) are usually disabled within this function or the SPC2 interrupt and the collect interrupt are explicitly disabled. When explicit disabling is used, **all** other interrupts (also those of lower priority) can extend this interrupt disable. SPC2 performance and thus AMPRO2 data flow may deteriorate.
unmask_int_asic_fct_ptr		Pointer to a user function which enables the SPC2 interrupt and collect interrupt again. Function type: *void (*unmask_int_asic_fct_ptr) (void);*
mask_int_global_fct_ptr		Pointer to user function which globally masks all interrupts Function type: *void (*mask_int_global_fct_ptr) (void);* Note: This interrupt disable is required for brief SPC2 lock times within AMPRO2 in addition to the "mask_int_asic_fct_ptr" disable. When this interrupt disable is extended by other causes, excessive SPC2 lock times can cause slower bus reaction times or bus collisions. A global interrupt disable is recommended for this reason. When explicit disabling is used, **all** interrupts (also those of lower priority) can extend this interrupt disable. When disabling by increasing the priority level to a certain value is used, all higher-priority interrupts may extend this interrupt disable. **Caution**: Since the interrupt disable via "mask_int_global_fct_ptr" is executed within the interrupt disable via "mask_int_asic_fct_ptr", an interrupt enable via "unmask_int_global_fct_ptr" may not cancel an interrupt disable via "mask_int_asic_fct_ptr".
unmask_int_global_fct_ptr		Pointer to user function which globally enables all interrupts again Function type: *void (*unmask_int_global_fct_ptr) (void);*
gen_int_os_collect_fct_ptr		Pointer to user function which generates the OS collect interrupt for AMPRO2 operating system integration. This function is not required unless AMPRO2 is being executed under an operating system environment (L2_TASK_IFA_OS und L2_CALL_IFA_OS). Function type: *void (*gen_int_os_collect_fct_ptr) (void);*
spc2_par.spc2_adr_valid		*FALSE*: The SPC2 address is set in config.h. *TRUE*: The SPC2 address is set in parameters "*spc2_adr_seg*" and "*spc2_adr_offset*". **Caution:** Only possible when the SPC2 is located in the FAR area ("*SPC2_ATTR_MEM_FAR*")
spc2_par.spc2_adr_seg		Segment of the SPC2 address (only valid for "*spc2_adr_valid*" = *TRUE*)
spc2_par.spc2_adr_offset		Offset of the SPC2 address (only valid for "*spc2_adr_valid*" = *TRUE*)
spc2_par.tok_err_limit		Threshold value for the number of non-plausible token telegrams per 256 consecutive token rotations When reached, the MAC generates the "*L2_MAC_RESET_LAS_USELESS*" MAC EVENT for active stations and goes to "listen token." Value range: 1 to 255
spc2_par.l4_hlen_var1	P	Layer-4 header length 1 in words: Value range: 1 to 16 (corresponding opcode mask *L2_L4_HLEN_VAR1*)
spc2_par.l4_hlen_var2	P	Layer-4 header length 2 in words: Value range: 1 to 16 (corresponding opcode mask *L2_L4_HLEN_VAR2*)
spc2_par.sap_max	P	Highest SAP number supported during receipt of indications (responder functionality) Value range: 0 to 63. The default SAP and SAP 0 are always supported. Note: When a request telegram is received on a SAP with a higher SAP number, the AMPRO2 SPC2 generates a negative acknowledgment for the bus with "no service activated" ("rs").

...

spc2_par.scb_base_ptr Pointer to system control block (SCB)
Note: The SCB must have a minimum length of *L2_LEN_SCB* (l2_user.h).

spc2_par.int_pol Polarity of the interrupt outputs: *L2_SPC2_INT_POL_LOW, L2_SPC2_INT_POL_HIGH*

spc2_par.int_config SPC2 interrupt mode: *L2_SPC2_NOT_SEP_INT*: All interrupts are on pin "INT-EV".
L2_SPC2_SEP_INT is not supported by the AMPRO2 SPC2.

spc2_par.rdy_config SPC2 ready signal mode: *L2_SPC2_NOT_EARLY_RDY*, *L2_SPC2_EARLY_RDY*

spc2_par.req_high_size Length of the "Request-Queue-High" SPC2 list in 8-byte multiples
Value range: 0: List not activated.
2 to 229: List is activated. Corresponds to telegrams with user data lengths of maximum of 6 to 246 bytes.

spc2_par.req_low_size Length of the "Request-Queue-Low" SPC2 list in 8-byte multiples
Value range: 0: List not activated.
2 to 229: List is activated. Corresponds to telegrams with user data lengths of maximum of 6 to 246 bytes.

spc2_par.reply_on_req_high_size Length of the "Reply-on-Request-High" SPC2 list in 8-byte multiples
Value range: 0: List not activated.
1 to 229: List is activated. Corresponds to telegrams with user data lengths of maximum of 6 to 246 bytes.

spc2_par.reply_on_req_low_size Length of the "Reply-on-Request-Low" SPC2 list in 8-byte multiples
Value range: 0: List not activated.
1 to 229: List is activated. Corresponds to telegrams with user data lengths of maximum of 6 to 246 bytes.

spc2_par.ind_size P Length of the "Indication-Queue" SPC2 list in 8-byte multiples
See also the *ind_buf_length_min* parameter.
Value range: 1 to 230 Corresponds to telegrams with user data lengths of maximum of 6 to 246 bytes.
Note: The following conditions must be met if the user wants to prevent the SPC2 from generating a "non resource" ("rr") acknowledgment for the bus.
- An indication resource must be available in the SAP for receipt of the telegram.
- The indication queue must be parameterized for at least 2 maximum-length telegrams (in accordance with PROFIBUS, a MAC indication is not triggered until the next telegram has been received correctly).
- The indication queue must have room for receipt of a new telegram. The SPC2 interrupt handler must empty the indication queue fast enough. We recommend assigning the SPC2 interrupt handler a high priority and not disabling it too long.
See also the *mask_int_asic_fct_ptr* parameter in the layer-2 init-block.

spc2_par.reply_on_ind_block1_number P Number of type-1 memory blocks for the "Reply-on-Indication" SPC2 list
Value range: 0 Parameter *spc2_par.reply_on_ind_block1_size* irrelevant
1 to 229 Parameter *spc2_par.reply_on_ind_block1_size* specifies the block size.

spc2_par.reply_on_ind_block1_size P Size of a type-1 memory block for the "Reply-on-Indication" SPC2 list in 8-byte multiples
Value range: 1 to 31 Corresponds to telegrams with user data lengths of maximum of 6 to 246 bytes

spc2_par.reply_on_ind_block2_number P Number of type-2 memory blocks for the "Reply-on-Indication" SPC2 list
Value range: 0 Parameter *spc2_par.reply_on_ind_block2_size* irrelevant
1 to 229 Parameter *spc2_par.reply_on_ind_block2_size* specifies the block size.

spc2_par.reply_on_ind_block2_size P Size of a type-2 memory block for the "Reply-on-Indication" SPC2 list in 8-byte multiples
Value range: 1 to 31 Corresponds to telegrams with user data lengths of maximum of 6 to 246 bytes

Baud Rate Table	
	AMPRO2 ASPC2 (48 MHz Clock)
L2_KBAUD_9_6	☞
L2_KBAUD_19_2	☞
L2_KBAUD_93_75	☞
L2_KBAUD_187_5	☞
L2_KBAUD_375	
L2_KBAUD_500	☞
L2_KBAUD_750	☞
L2_MBAUD_1_5	☞
L2_MBAUD_2_625	
L2_MBAUD_3	☞
L2_MBAUD_3_5	
L2_MBAUD_4	☞
L2_MBAUD_5_25	
L2_MBAUD_6	☞
L2_MBAUD_10_5	
L2_MBAUD_12	☞

FLC_FMA_MAC_RESET CONFIRMATION

next_blk_ptr	Undefined
prev_blk_ptr	Undefined
opcode	*L2_FMA_CONFIRM*
subsystem	Unchanged in comparison to REQUEST
id_ptr	Unchanged in comparison to REQUEST
as.fma.service_code	*L2_FLC_FMA_MAC_RESET* (unchanged in comparison to REQUEST)
as.fma.status	The following values for *status* contain the status of the confirmation in the 8 bits. See section 1.1.6.7. - *L2_STATUS_OK*:Positive acknowledgment. FLC_FMA_MAC_RESET was executed. - *L2_STATUS_IV*: Negative acknowledgment. Invalid parameter in REQUEST. - Parameter error in the layer-2 init-block or in the layer-2 bus parameter block
as.fma.ptr1	Unchanged in comparison to REQUEST. Unchanged layer-2 init-block returned.
as.fma.ptr2	Unchanged in comparison to REQUEST. Unchanged layer-2 bus parameter block returned.
as.fma.length	Unchanged in comparison to REQUEST
as.fma.sap_nr	Unchanged in comparison to REQUEST
as.fma. ...	Unchanged in comparison to REQUEST

3.9 General Return Value

3.9.1 "resp_status"

Value range for "as.flc.resp_status" for MAC confirmations (4-bit evaluation)

Bit 7 6 5 4	3 2 1 0			
(. w . .)	0 0 0 0	(0x00)	*L2_STATUS_OK*	(Bits 4 to 7 must be masked.) more information in bits 4 to 7 (short ack. SC yes/no, ...)
(0 w . .)	0 0 0 1	(0x01)	*L2_STATUS_UE*	(Bits 4 to 7 must be masked.)
(0 w . .)	0 0 1 0	(0x02)	*L2_STATUS_RR*	(Bits 4 to 7 must be masked.)
(0 w . .)	0 0 1 1	(0x03)	*L2_STATUS_RS*	(Bits 4 to 7 must be masked.)
(0 0 0 0)	0 1 0 1	(0x05)	*L2_STATUS_WD*	(Bits 4 to 7 must be masked.)
(0 w . .)	1 0 0 0	(0x08)	*L2_STATUS_DL*	(Bits 4 to 7 must be masked.)
(0 w . .)	1 0 0 1	(0x09)	*L2_STATUS_NR*	(Bits 4 to 7 must be masked.) Remark 1)
(0 w . .)	1 0 1 0	(0x0A)	*L2_STATUS_DH*	(Bits 4 to 7 must be masked.)
(0 0 0 0)	1 0 1 1	(0x0B)	*L2_STATUS_IV*	(Bits 4 to 7 must be masked.)
(0 w . .)	1 1 0 0	(0x0C)	*L2_STATUS_RDL*	(Bits 4 to 7 must be masked.) Remark 1)
(0 w . .)	1 1 0 1	(0x0D)	*L2_STATUS_RDH*	(Bits 4 to 7 must be masked.) Remark 1)
(1 0 0 0)	1 1 1 0	(0x0E)	*L2_STATUS_IL*	(Bits 4 to 7 must be masked.)
(1 w . .)	1 1 1 1	(0x0F)	*L2_STATUS_NA*	(Bits 4 to 7 must be masked.): more information in bits 4 to 7

0 w . .	0 0 0 0	(0x00)	*L2_STATUS_OK*:	No short ack. in acc. w. PROFIBUS
1 w 0 0	0 0 0 0	(0x80)	*L2_STATUS_SC + L2_STATUS_OK*:	Short ack. (SC) in acc. w. PROFIBUS
1 w 0 1	1 1 1 1	(0x9F)	*L2_STATUS_NA_TIMEOUT*:	Timeout occurred (no reaction or no plausible reaction from remote station
1 w 1 0	1 1 1 1	(0xAF)	*L2_STATUS_NA_DOUBLE_TOKEN*:	Double token occurred (no plausible reaction from remote station).
1 w 1 1	1 1 1 1	(0xBF)	*L2_STATUS_NA_BUFFER_ERROR*:	Buffer error occurred while receiving a response telegram. Receiving buffer does not exist on local station or is too small, or remote station sent response data by mistake.

In accordance with PROFIBUS, bits 4 and 5 contain the **FDL** status of the remote station for the states *L2_STATUS_OK* (**not** for short acknowledgment SC, SDN and NOP), *L2_STATUS_UE*, *L2_STATUS_RR*, *L2_STATUS_RS*, *L2_STATUS_DL*, *L2_STATUS_NR*, *L2_STATUS_DH*, *L2_STATUS_RDL* and *L2_STATUS_RDH* .

. . 0 0	. . . .	(0x00)	*L2_STATION_PASSIVE*:	Passive station
. . 0 1	. . . .	(0x10)	*L2_STATION_ACTIVE_NOT_READY*:	Active station not ready Remark 1)
. . 1 0	. . . .	(0x20)	*L2_STATION_ACTIVE_READY*:	Active station ready for logical token ring
. . 1 1	. . . .	(0x30)	*L2_STATION_ACTIVE*:	Active station in logical token ring

In addition, the AMPRO2 ASPC2 indicates in bit 6 whether a **telegram repeat** was or was not performed in accordance with PROFIBUS during execution of this MAC job. This bit is set when a repeat occurred. This bit is always 0 for all other AMPRO2 versions.

☞ In a MAC repeat job, the repeat bit is only valid for the last job performed (cycle).

. 1 . .	. . . .	(0x40)	*L2_STATUS_RETRY*:	Telegram repeat in accordance with PROFIBUS

Examples:

1 0 1 1	0 0 0 0	*L2_STATUS_SC + L2_STATION_ACTIVE + L2_STATUS_OK*
0 0 1 1	0 0 1 1	*L2_STATION_ACTIVE + L2_STATUS_RS*
0 1 0 0	1 0 0 0	*L2_STATUS_RETRY + L2_STATION_PASSIVE + L2_STATUS_DL*

Remarks: 1) These states do not occur when the remote station contains an AMPRO2 implementation.

Additional value range for "as.flc.resp_status" and "as.fma.status" (8-bit evaluation):

Bit 7 6 5 4	3 2 1 0			
0 0 0 0	0 0 0 0	(0x00)	*L2_STATUS_OK*	(All 8 bits are valid.)
0 0 0 0	0 1 0 0	(0x04)	*L2_STATUS_LO*	(All 8 bits are valid.)
0 0 0 0	0 1 0 1	(0x05)	*L2_STATUS_LL*	(All 8 bits are valid.)
0 0 0 0	0 1 1 0	(0x06)	*L2_STATUS_NO*	(All 8 bits are valid.)
0 0 0 0	0 1 1 1	(0x07)	*L2_STATUS_LR*	(All 8 bits are valid.)
0 0 0 0	1 0 1 1	(0x0B)	*L2_STATUS_IV*	(All 8 bits are valid.)
0 0 0 0	1 1 1 1	(0x0F)	*L2_STATUS_TO*	(All 8 bits are valid.)
0 0 0 0	0 0 0 0	(0x00)	*L2_STATUS_OK_PROCESSED*	((All 8 bits are valid.)
0 0 0 0	0 1 0 1	(0x05)	*L2_STATUS_OK_NOT_PROCESSED*	(All 8 bits are valid.)

3.10 Error Outputs on AMPRO2

☞ Function type: *void user_error_fct (UBYTE error_code, void PTR_ATTR *error_ptr);*
Error codes: See below.
Error ptr: The error pointer is either 0 or points to the application block which was being processed by AMPRO2 when the error occurred.

☞ See also description of the *user_error_fct_ptr* parameter in the layer-2 init-block. (FLC_FMA_MAC_RESET service)

Error codes for the AMPRO2 ASPC2:

(0x50)	*L2_INTERNAL_OPCODE_ERROR*	Opcode error on AMPRO2
(0x51)	*L2_CON_IND_OPCODE_ERROR*	Opcode error on the L2 con-ind handler
(0x53)	*L2_ASPC2_STEP_ERROR_TOO_OLD*	ASPC2 STEP error. This STEP is no longer supported by this AMPRO2. (Check "ASPC2_GREATER_EQUAL_STEP_x" in config.h.)
(0x54)	*L2_ASPC2_STEP_ERROR_TOO_NEW*	ASPC2 STEP error. This STEP is no longer supported by this AMPRO2. (A more recent AMPRO2 may need to be obtained.)

4 AMPRO2 CBF Distributor

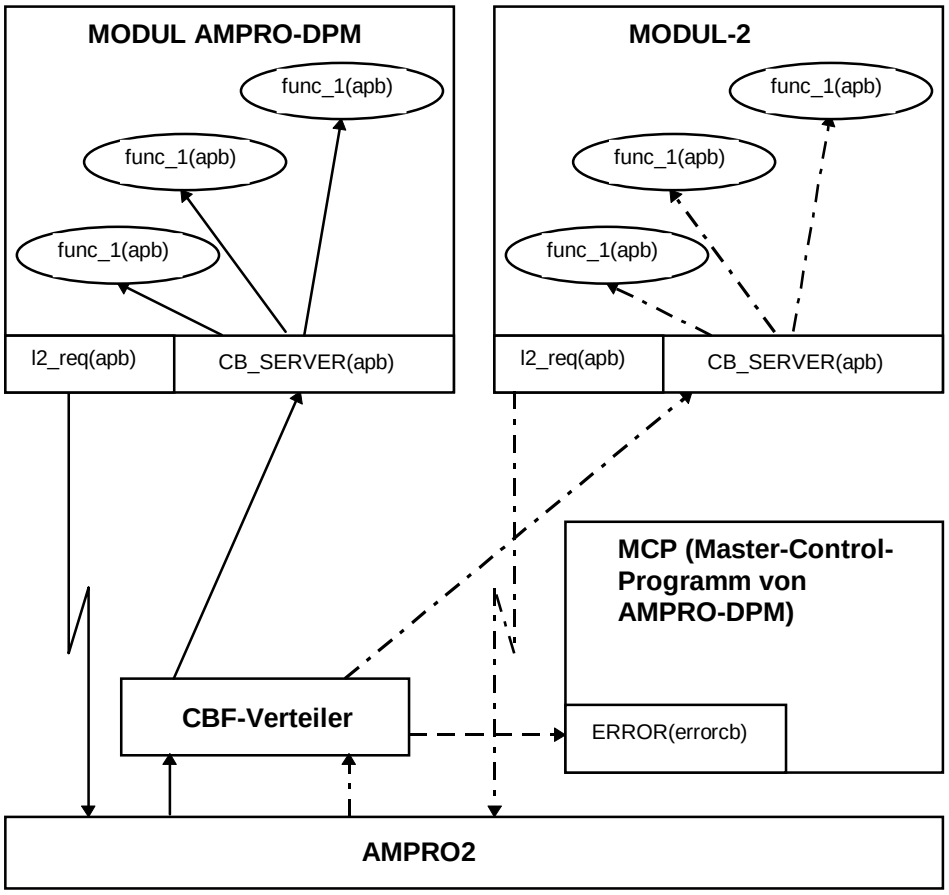
4.1 General

Since AMPRO2 does not yet have l2_open() functionality, a call back function distributor (i.e., CBF distributor) has been implemented. This CBF distributor must be available to all other AMPRO2 users so that the CBFs can be sent to the appropriate users.

4.1.1 Description of the Procedure

Each module required by AMPRO2 issues service requests as the requester with the "l2_req(apb_ptr)" call to AMPRO2, or expects confirmation as the responder from AMPRO2. The confirmation or response is reported via call back functions to the responder or requester. The appropriate job block (apb_ptr) is transferred to the responder/requester with the confirmation/response. The job can now be evaluated by the applicable module based on the job block.

The requester sends a l2_req() to AMPRO2. AMPRO2 executes the job and sends the response to the CBF distributor. The distributor uses the job block (apb_ptr->subsystem) to determine the applicable module and calls the CBF server (CB_SERVER(apb_ptr) of the module with the reference of the job block. The final call back function is called on the CBF server based on the job block (apb_ptr->subsystem).



Module structure of the CBF distributor/server concept based on the AMPRO2-DPM

4.1.2 Definitions

The following files are required for the CBF distributor.

- SYS_CBD.H // Contains the required definitions
- SYS_CBD.C // Contains the CBF distributor

The CBF distributor internally calls an error function when errors occur. This function is reported to the CBF distributor and must correspond to the error function of the AMPRO2-DPM.

4.1.2.1 CBF Distributor Structure

The CBF distributor requires the following structure.

```
struct  srv_def
{
   /* CBF server function pointers */

   void   SYS_CBD_FUNC_ATTR error   (ERRCB);
   void   SYS_CBD_FUNC_ATTR server_function_1 (L2_APB_PTR);
   void   SYS_CBD_FUNC_ATTR server_function_2 (L2_APB_PTR);
   void   SYS_CBD_FUNC_ATTR server_function_3 (L2_APB_PTR);
   void   SYS_CBD_FUNC_ATTR server_function_4 (L2_APB_PTR);

};
```

The size of the structure must be adapted to the AMPRO2 users. The structure is currently designed for 4 different modules. When more modules are required, the server_function_x() entries must be expanded.

4.1.2.2 CBF Distributor Function

The CBF distributor is defined as follows in the SYS_CBD.C file.

```
void L2_CALL_BACK_CODE_ATTR  ampro2_cbf_distributor   (void)
```

4.1.2.3 CBF Server Function

The CBF server functions of the individual modules or the CBF distributor must be defined as shown below.

```
void SYS_CBD_FUNC_ATTR server_function_X (L2_APB_PTR);
```

Since the CBF server function is inserted as a pointer in the CBF distributor structure, any function name can be selected.

4.1.2.4 CBF Error Function

The CBF distributor also requires an error() function. This error() function is also inserted in the CBF distributor structure. The ERRCB structure is not described here. For details, see the specifications of the AMPRO-DPM.

```
void  SYS_CBD_FUNC_ATTR error (ERRCB)
```

4.2 Implementation

Each module required by AMPRO2 must have a way of receiving the call back functions of AMPRO2. The number of required CBF functions per module and the number of AMPRO2 users must be known to the CBF distributor.

To make assignment of the CBF functions to the AMPRO2 users simple, use the following organization.

CBF user 1 is assigned function identifier 1 for the CBF distributor.
CBF user 2 is assigned function identifier 2 for the CBF distributor.
...
CBF user x is assigned function identifier x for the CBF distributor.

When more than the 4 AMPRO2 users currently provided for exist, the concept of the CBF distributor must be expanded by the user.

The SYS_CBD_FUNC_ATTR attribute is defined for the function attributes in "fw_defma.h". Setting of this attribute depends on the hardware and must be set by the user in accordance with his particular requirements.

4.2.1 Number of AMPRO2 CBFs Per User

The number of CBF functions per CBF server must be specified for the CBF distributor. The following specifications have been used for our example.
- For CBF server 1: 70 CBF functions
- For CBF server 2: 20 CBF functions
- For CBF server 3: 20 CBF functions
- For CBF server 4: 20 CBF functions

Definition of the CBF areas for the individual AMPRO2 users in SYS_CBD.H

```
/*+-------------------------------------------------------------------------------------+*/
/*¦ Note: for the IM308C-Hardware   the FUNC1-define is used by the dpm_l2_cb_server,   ¦*/
/*¦                                 the FUNC2-define is used by the dpx1_l2_cb_server,  ¦*/
/*¦                                 the FUNC3-define is used by the dpx2_l2_cb_server,  ¦*/
/*¦                                 the FUNC4-define is used by the dps_l2_cb_server,   ¦*/
/*¦ This general layout can used by every AMPRO2-CBF-USER                               ¦*/
/*+-------------------------------------------------------------------------------------+*/


    #define SRV_FUNC1_CBF_NR            70 /* numbers of cbfs for cbf-server-function1
    #define SRV_FUNC2_CBF_NR            20 /* numbers of cbfs for cbf-server-function2
    #define SRV_FUNC3_CBF_NR            20 /* numbers of cbfs for cbf-server-function3
    #define SRV_FUNC4_CBF_NR            20 /* numbers of cbfs for cbf-server-function4

    #define SRV_FUNC1_MIN_NR            0  /* function_1 begins with subsystem-n

    #define SRV_FUNC1_MAX_NR            SRV_FUNC1_MIN_NR + SRV_FUNC1_CBF_NR - 1

    #define SRV_FUNC2_MIN_NR            SRV_FUNC1_MAX_NR
    #define SRV_FUNC2_MAX_NR            SRV_FUNC2_MIN_NR + SRV_FUNC2_CBF_NR - 1

    #define SRV_FUNC3_MIN_NR            SRV_FUNC2_MAX_NR
    #define SRV_FUNC3_MAX_NR            SRV_FUNC3_MIN_NR + SRV_FUNC3_CBF_NR - 1

    #define SRV_FUNC4_MIN_NR            SRV_FUNC3_MAX_NR
    #define SRV_FUNC4_MAX_NR            SRV_FUNC4_MIN_NR + SRV_FUNC4_CBF_NR - 1
```

4.2.2 CBF Server Functions

Each user must provide a function which calls the applicable CBF of the module (i.e., the CBF server). It is practical to identify this CBF server when the corresponding AMPRO2 user is initialized (e.g., in the AMPRO2 user's own "open()" function) and enter it in the CBF distributor structure.

The CBF server receives a pointer to the application block (APB) as transfer parameter. The APB contains the "apb->subsystem" entry in which the original identifier of the CBF is entered. The CBF can now call the appropriate function using this identifier.

We recommend using a pointer array for implementation of the CBF server. Based on "apb->subsystems", a jump is then made within the CBF server to the array and the function pointer to which the array entry is assigned is called.

Definition the AMPRO-DPM CBF Server in AMPRO-DPM

```
/*+----------------------------------------------------------------------------+*/
/*¦ function:       dpm_l2_cb_server                                         ¦*/
/*¦                                                                          ¦*/
/*¦ duty:           At the moment AMPRO2 is not able to call USER-CBF's, so  ¦*/
/*¦                 the USER has to call the CBF's by himself. While this    ¦*/
/*¦                 ability is missing, the following function calls all     ¦*/
/*¦                 CBF's for AMPRO-DPM.                                     ¦*/
/*¦                                                                          ¦*/
/*¦ parameters:     L2_APB_PTR                                               ¦*/
/*¦                                                                          ¦*/
/*¦ return value:   none                                                     ¦*/
/*+----------------------------------------------------------------------------+*/


DPM_EXTERN_SMCO void SYS_CBD_FUNC_ATTR dpm_l2_cb_server (L2_APB_PTR apb_ptr)
{
        (*_dpm.cbf.arr[apb_ptr->subsystem].func_ptr) (apb_ptr);
}
```

Definition of the CBF structure in the AMPRO2-DPM for its CBF server

```
struct  cbf_def
{
    Unsigned8   func_code;
    Unsigned8   res;
    void        func_ptr_name__(L2_CALL_BACK_CODE_ATTR,L2_APB_PTR);
};

struct  cbf_str_def
{
    struct  cbf_def     diag1;
    struct  cbf_def     prm;
    struct  cbf_def     cfg;
    struct  cbf_def     diag2;
    struct  cbf_def     data;
    struct  cbf_def     prm_unlock;
...
};
```


4.2.3 Initialization of the AMPRO2 CBF Distributor

The "_srv" CBF distributor structure must be initialized by a higher-level instance. The CBF server functions and an error function are entered as pointers in this structure. The structure must be initialized by AMPRO2 before startup. CBF server functions which are not used must be initialized with ZERO. The error function is called by the CBF distributor when an error occurs. FW_ERROR.H contains the error definitions of the error function.

In addition to initializing the structure, the actual CBF distributor ("ampro2_cbf_distributor(void)") must be entered in ASIC event.
The required AMPRO2 interrupt handler has already been called internally before ampro2_cbf_distributor(void).

```
/* enter the "error-function" for the ampro2_cbf_distributor() */
_srv.error = system_error_function;
```

Initialization of the CBF server for the AMPRO-DPM after the dpm_open() function

```
/* get the dpm_l2_server-function and enter it in the _srv-struct */
_srv.server_function_1  = mcp. dpm_ptr-> dpm_l2_cb_server;
```

5 DPM Interface

5.1 Introduction

5.1.1 Communication Model

Communication of the individual SW elements with the AMPRO-DPM and with AMPRO2 is handled by function calls and ***call back functions*** (i.e., CBF). This technique permits hardware-dependent program sections to be relocated easily, thus making AMPRO2 and AMPRO-DPM independent of the hardware. An otherwise essential mailbox system can also be omitted along with the time required for transmission and coordination (e.g., by an operating system).

Initialization of this service is triggered by the user with normal function calls. The user uses an XXX_OPEN call to obtain the addresses of these functions from the partner beforehand. All AMPRO-DPM functions which are relevant to the USER are explained in detail in the section on AMPRO-DPM functions.

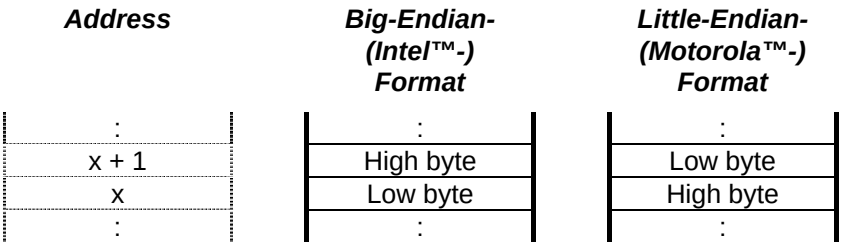
In contrast, call back functions must be supplied by the user. AMPRO2 or AMPRO-DPM requires the address of this function before it can call a CBF. Their transfer by the USER is performed in the job blocks described below. Pointers to the job blocks are usually the transfer parameters of the normal functions. When an event occurs in a SW package (e.g., AMPRO-DPM) which also pertains to the other SW package (e.g., that of the USER), a previously specified function of the other package is called. Page 66 of the section on sequence charts shows a flow chart of the procedure.
Since CBFs are usually executed at the interrupt level, their execution times and code should be kept as short as possible. This function must still be provided even when a CBF is not required by the destination system. Its body can consist of the "return" command only (i.e., dummy function). The same applies to all normal functions described here (detailed description starting on page 86) and CBFs (detailed description starting on page 127).
Communication with the CBF permits recursive function calls. For example, when the AMPRO-DPM calls a function of the USER, the USER could call an AMPRO-DPM function before exiting the first function. Such recursive calls are only possible under certain conditions. They may **never be used** for the AMPRO-DPM. AMPRO2 may only execute a maximum of one additional AMPRO2 job from a call back function. See the specifications of AMPRO2 for prerequisites applicable to the use of recursive calls by AMPRO2.
All functions of the AMPRO-DPM are **non-reentrant** (i.e., the user may not start the same function again as long as a function call is running). The AMPRO-DPM usually performs a plausibility check on this. An AMPRO-DPM function may also not be called from a CBF. When the USER wants to use an AMPRO-DPM function in reaction to an AMPRO-DPM CBF, he must set a flag within the CBF, conclude the CBF, and then start the required function based on the flag. See also section on call structure for the USER on page 81.
Most CBFs of the USER must be able to be called within the USER call. This procedure is described in more detail in the description of the CBFs starting on page 127.

The AMPRO-DPM does not take action on its own initiative. It requires external triggering to process the required software sections. This trigger is provided by the USER for new jobs or for time control of the USER. Cyclic processing of already issued jobs is ensured by return messages from AMPRO2 to AMPRO-DPM.

5.1.2 Definitions

5.1.2.1 Sequence of Bytes of a Word

There are two ways to organize the two bytes of a word.



Unfortunately, both techniques must also be used in this system depending on the particular definition. For example, all words which are sent via PROFIBUS-DP must be transferred in Little-Endian format. The required word format is pointed out separately in some important instances.

5.1.2.2 Use of the ISO/OSI Reference Model

Of the 7 layers of the ISO/OSI reference model, layers 1 and 2 of the PROFIBUS standard are defined for a PROFIBUS system. On AMPRO2, layer 2 is also divided into two "half layers" so that the jobs of the individual layers can be better described. The following figure shows the divisions and designations of the layers. The abbreviations used in the figure are also used for the applicable AMPRO2 services. For example, the AMPRO2 command for a general reset of the interface is FLC_FMA_MAC_RESET.

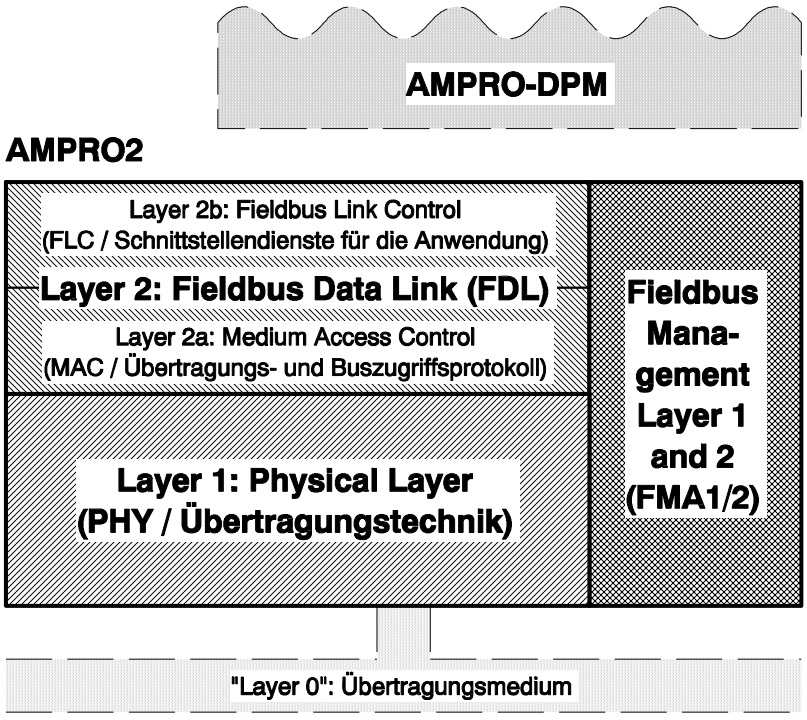


Figure 2: Use of the ISO/OSI reference model for AMPRO2

5.1.2.3 Inputs/Outputs

The term *input* or *output* as used in these specifications always refers to the DP master. Data which are transferred from a DP slave to the DP master are inputs. Data which are transferred from the DP master to the DP slave are outputs. The following figure illustrates these designations.

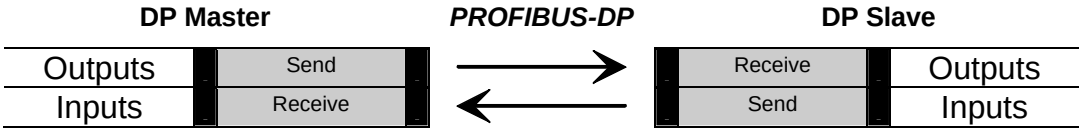


Figure 3: Direction of data transmission for inputs and outputs

5.1.2.4 Types of Variables

The following globally accessible definitions exist for the variable types used in these specifications in accordance with the PROFIBUS-DP standard.

Designation	***Type***	***Length***	***Sign***
Octet		8 bits	-
Unsigned8	Integer	8 bits / 1 octet	Without sign
Unsigned16	Integer	16 bits / 2 octets	Without sign
Unsigned32	Integer	32 bits / 4 octets	Without sign
Signed8	Integer	8 bits / 1 octet	With sign
Signed16	Integer	16 bits / 2 octets	With sign
Signed32	Integer	32 bits / 4 octets	With sign
Boolean	Integer	8 bits / 1 octet	Without sign
Bitfield	Integer	-	Without sign

An octet is always an 8-bit value for which no sign is defined. All other data types are made up of one or more octets.
Boolean-type variables can only assume one of the two values DP_TRUE or DP_FALSE. See also the section on coding rules for the definitions of boolean values starting on page 154. The size of the integer for the type bitfield is not constant since it corresponds to the natural integer size of the processor and the compiler. In accordance with ANSI-C regulations, it is defined as "unsigned int" without the modifier "long" or "short" so that the compiler can use the type most suitable for it. The variable type "bit" (1-bit integer without sign) is sometimes used in these specifications. This is not a separate type. A bit is addressed via a bitfield (format: bitfield) or via binary integer operations (format: unsigned16). Details are provided at appropriate points.
In addition to these basic types, the octet-string and visible-string types are mentioned in the DP standard. Since these types are not explicitly used by the AMPRO-DPM, they have not been defined separately.

5.1.2.5 Data/Coding Areas

5.1.2.5.1 Overview

Several different memory attributes (e.g., near, far, huge, etc.) must be assigned for the data and codes based on the individual location of the particular memory areas and processor type. All attributes are provided with definitions so that the attributes can be changed and remain independent of each other. All pointer accesses and all function calls must be modified with attribute definitions even when the selected attribute is a default setting. In principle, each firmware section can be used with another attribute internally and for communication with other sections. These have already been defined for AMPRO2 in the "\COMMON\CONFIG.H" file. In addition, the specifications described in the following sections have been made for AMPRO2.

The USER must specify the area modifiers when the system is configured. This information can then be used to generate the user's own AMPRO2-DPM library especially customized to this one system. When performing any further system generations, the USER must adhere to the agreements previously made or request a new AMPRO2-DPM library.

The following figure shows all required attribute definitions and illustrates their direction of action.

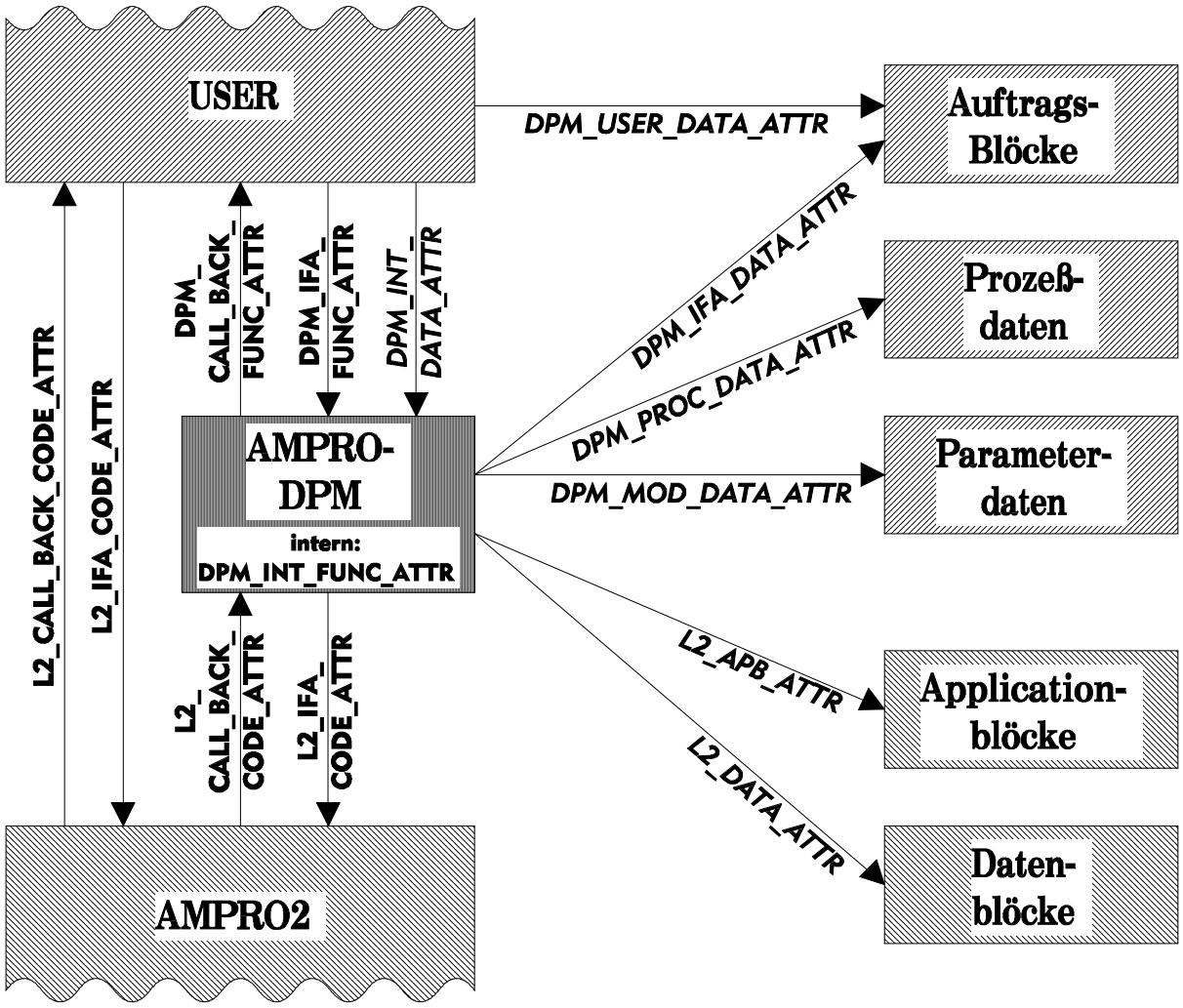


Figure 4: Illustration of area attributes

5.1.2.5.2 Attributes for AMPRO2

The mnemonic "L2_..." contains all definitions which are relevant to the AMPRO2. The two AMPRO2 function attributes apply equally to the USER and AMPRO-DPM. In addition to the area for internal data, the AMPRO2 data area is divided into two parts (i.e., the area for application blocks and the area for the blocks of data to be transferred).

L2_IFA_CODE_ATTR	Attribute for AMPRO2 functions called by the user (i.e., AMPRO-DPM and the USER)
L2_CALL_BACK_CODE_ATTR	Attribute for user functions called by AMPRO2
L2_APB_ATTR	Attribute for the AMPRO2 memory area for application blocks
L2_DATA_ATTR	Attribute for the AMPRO2 memory area for data blocks

5.1.2.5.3 Attributes for AMPRO-DPM

Similar to AMPRO2, the mnemonic "DPM_..." is assigned to all areas for communication between AMPRO-DPM and the USER. The following definitions are used.

DPM_IFA_FUNC_ATTR	Attribute for the AMPRO-DPM functions called by the USER
DPM_CALL_BACK_FUNC_ATTR	Attribute for the USER functions called by AMPRO-DPM
DPM_INT_FUNC_ATTR	Attribute for AMPRO-DPM functions only called by AMPRO-DPM itself (i.e., internal functions)
DPM_IFA_DATA_ATTR	Attribute for the AMPRO-DPM data area (i.e., slave control and job blocks) for the USER
DPM_INT_DATA_ATTR	Attribute for the internal AMPRO-DPM data area (structure DPM). The USER receives a pointer to this area after calling the "dpm_open ()" function. See page 86 ff.

5.1.2.5.4 Other Attributes

In addition to the mnemonics for the individual firmware parts, mnemonics have been introduced for certain memory areas which cannot be associated directly with a package. These memory areas are the area for process data (i.e., inputs, outputs and diagnoses) and the area for parameter data records.

DPM_PROC_DATA_ATTR	Attribute for the process data area (i.e., inputs, outputs and diagnoses)
DPM_MOD_DATA_ATTR	Attribute for the parameter data records

The USER can also make entries for identification of the job blocks described below. These have an separately defined, internal USER data attribute.

DPM_USER_DATA_ATTR	Attribute for the USER data area

5.1.2.6 Direction of Access for Processing the Job Blocks

All job blocks described in these specifications have four columns to the right showing the accessing direction of the individual fields of a block. The two USER columns apply to the user. The two other columns apply to AMPRO-DPM. The meaning of the entries in the USER columns is shown below.

Entry	***Meaning***
No entry	This field cannot be read or write-accessed by the USER. These fields are reserved for AMPRO-DPM.
"X" in L column only	This field can only be read-accessed by the USER (e.g., for status messages and so on), but not write-accessed.
"X" in both columns	This field must be write-accessed by the USER. The USER may also monitor his entries.
"X" in S column only	Not possible. Would serve no useful purpose anyway.

5.1.2.7 Sequence Charts

Sequence charts have been prepared for most commands. These charts illustrate the principle of procedure of a function and are provided for better comprehension of the internal processes and the resulting reactions of AMPRO-DPM. The following figure shows such a sequence chart using a call back function as an example.

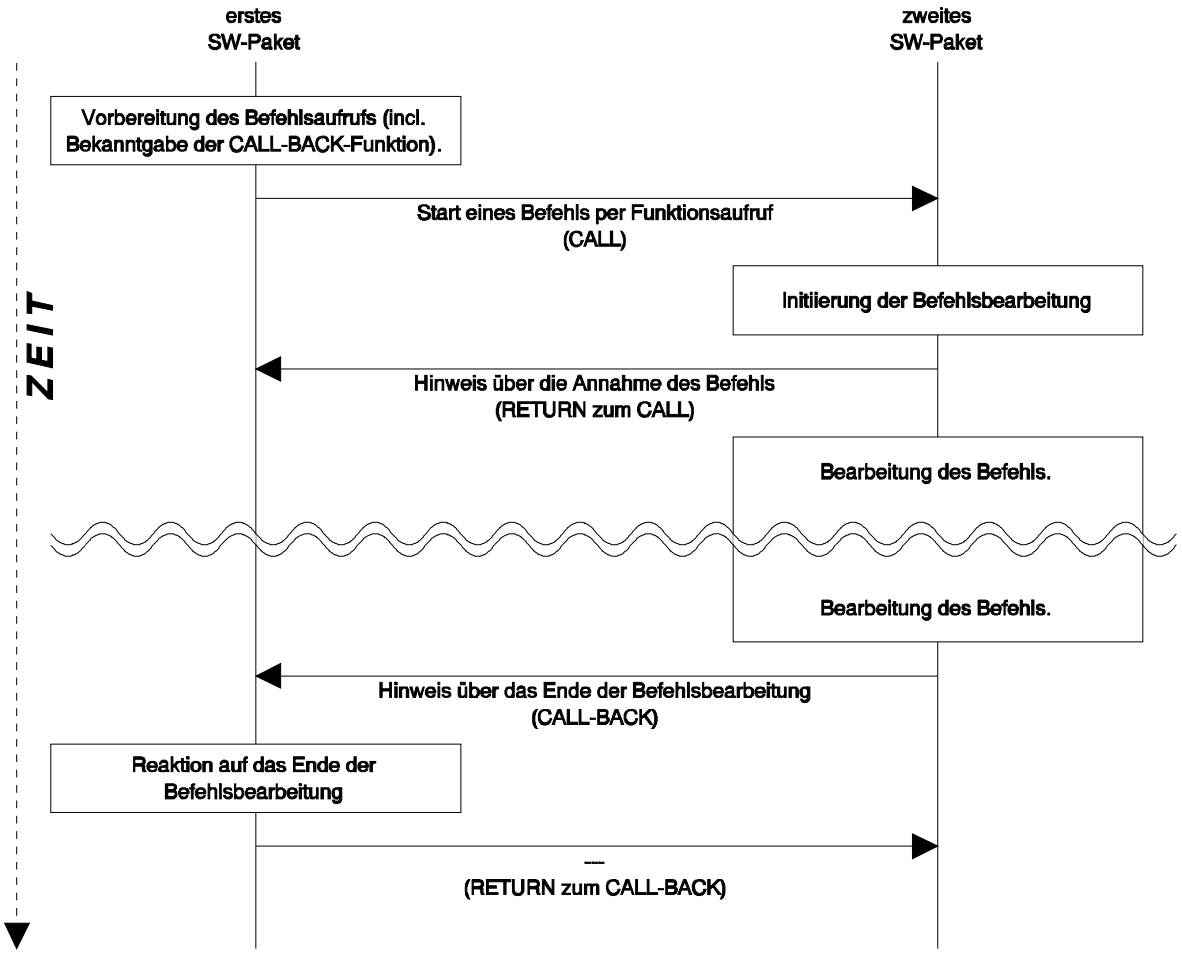


Figure 5: A sequence chart

The function designations used in the sequence charts do not show the precise names of the individually called functions. Collective designations and short designations have been used to provide a clear overview and not confuse you with details.
For an in-depth description of the function procedures and the correct designations, see the SDL diagrams in part 2 of this documentation or the PROFIBUS-DP standard.

5.1.3 Structure of AMPRO-DPM

AMPRO-DPM consists of three parts which will now be described in more detail.

- DPM-USIF DPM **Us**er **I**nter**f**ace
- DPM-SLSM DPM **Sl**ave **S**tate **M**achine
- DPM-DATR DPM **Da**ta **Tr**ansfer

5.1.3.1 DPM-USIF (AMPRO-DPM User Interface)

The DPM-USIF is started by the USER with the AMPRO-DPM functions. See below. Using the USER's parameters, the DPM-USIF generates all required job application blocks and internal management structures including the required memory allocation, and transfers the first job to AMPRO2 (e.g., for the first diagnostic request after the "add_slave ()" AMPRO-DPM function is called). After receiving the response, AMPRO2 returns the application blocks to the DPM-SLSM and not the DPM-USIF.
As already mentioned, functions which are not required can be programmed as dummy functions (i.e., only one "return" as function body). The next few sections provide a detailed description of the individual functions and the required transfer parameters.

5.1.3.1.1 List of the AMPRO-DPM Functions

The DPM-USIF offers the user the following functions.

dpm_open:	Registers the USER as user of AMPRO-DPM and permits the user access to all of the following functions. See description starting on page 86.
init:	Initializes the internal management structures and allocates appropriate resources. See description starting on page 88.
add_slave:	Adds a slave to AMPRO-DPM management, allocates the appropriate resources and, if possible, starts data communication with the slave. See description starting on page 96.
withdraw_slave:	Concludes data communication with a slave, releases the related resources and removes the slave from AMPRO-DPM management. See description starting on page 100.
restart_slave:	If possible, restarts data communication with a slave which has exited the DATA or DIAG2_STATUS state in master operating mode AUTOSTOP. See description starting on page 103.
set_master_mode:	Performs the change in state of the master and handles the slaves accordingly. See description starting on page 104.
set_slave_mode:	Sends a global control command to one, several or all slave groups assigned to the master, or to one specific slave.

See description starting on page 107.

set_slave_address:	Sets the PROFIBUS station address of a slave whose address can be changed. See description starting on page 113.
mark_cycle:	Reports a complete slave polling cycle to the USER. This service is used to determine whether all slaves have received current data. See description starting on page 117.
close:	Logs off the USER as user of AMPRO-DPM. See description starting on page 126.

In addition to these basic functions, the DPM-USIF offers functions for processing long consistency and Buffered_Mode.

input_update:	Starts processing for receipt of new input data. See description starting on page 118.
output_update:	Starts processing for transmission of new output data. See description starting on page 121.
consistency_update:	Update cycle for updating the input buffer data of slaves with long consistency or Buffered_Mode slaves. See description starting on page 123.

A timer is required for processing the slaves. This timer ***must*** be provided by the USER. Correct processing of the standard DP protocols is not possible without this timer. The following function is part of the timer processing system. The function must be called cyclically by the user.

timer_expired:	Executes actions required after the timer provided by the USER expires. See description starting on page 124.

Some parts of the communication are handled by CBFs and are interrupt-controlled. To be able to call these CBFs within the interrupt routine, AMPRO-DPM provides the user with a function distributor.

dpm_l2_cb_server:	Calls the AMPRO-DPM CBFs during an interrupt routine. See description starting on page 127.

5.1.3.1.2 List of the CBFs of the USER

The USER should also provide functions for AMPRO-DPM. These are used to inform the USER of important occurrences on AMPRO-DPM. They are also indispensable for maintenance of data communication.

init_done:	Response message for calling the "init ()" function. See description starting on page 128.
state_report:	Operating state indication for a slave. See description starting on page 128.
withdraw_slave_done:	Response message for calling the "withdraw_slave ()" function. See description starting on page137.
set_master_mode_done:	Response message for calling the "set_master_mode ()" function. See description starting on page 136.
set_slave_mode_done:	Response message for calling the "set_slave_mode ()" function. See description starting on page 138.
set_slave_address_done:	Response message for calling the "set_slave_address ()" function. See description starting on page 138.

mark_cycle_done:	Response message for the "mark_cycle ()" job. See description starting on page 139.
set_timer:	The USER must start the timer reserved for AMPRO-DPM. See description starting on page 140.
bus_accessible:	Note on functionality of the bus (e.g., when a short circuit occurs). See description starting on page 140.
error:	An error which AMPRO-DPM is unable to handle has occurred. This error corresponds to a system crash. The USER must provide a reaction (e.g., software reset). See description starting on page 143.

The following CBFs are required to ensure data consistency.

write_inp_data_to_pda:	The USER must copy inputs from a buffer to the process data area while the consistency disable is activated. See description starting on page141.
read_outp_data_from_pda:	The USER must copy outputs from the process data area to a buffer while the consistency disable is activated. See description starting on page141.
write_diag_data_to_pda:	The USER must copy diagnostic data from a buffer to the process data area while the consistency disable is activated. See description starting on page142.
clear_cons_input_data:	The USER must clear the outputs in the process data area . while the consistency disable is activated. See description starting on page 142.
consistency_update_done:	The USER receives a message indicating that the update cycle for the inputs of the slaves has been performed with long consistency or Buffered-Mode. See description starting on page 143.

The following CBFs are required for certain S7_Slaves for which the configuration is requested after startup.

copy_s7_get_cfg_data:	The USER receives a pointer-pointer to the buffer with the current configuration data which the user must check. See description starting on page 141.

In addition to these functions, AMPRO-DPM must be temporarily able to ensure that it will not be interrupted by an ASIC interrupt and thus by an AMPRO2 function. The functions for disabling and enabling the ASIC interrupt must already be supplied by the user for AMPRO2. See page 79 ff.

asic_int_disable:	Disable the ASIC interrupt. See description starting on page 143.
asic_int_enable:	Enable the ASIC interrupt. See description starting on page 143.

5.1.3.2 DPM-SLSM (AMPRO-DPM Slave State Machine)

The DPM-SLSM is started by AMPRO2 when it receives a job block with a confirmation. The originator of this job was the DPM-SLSM itself or the DPM-SLSM. The DPM-SLSM evaluates the block and handles the related slave in accordance with the PROFIBUS-DP standard, or, when a DP Siemens slave is involved, in accordance with ET 200 communication specifications. AMPRO-DPM enters the next status which the slave assumes in the slave control block (SLCB), transfers to AMPRO-DP the application block required for the change in state and waits for this block to be returned.
For some changes in state or for certain changes within a state in the state machine, AMPRO-DPM calls a call back function of the user whose address was previously provided to AMPRO-DPM by the user. See description of the "state_report ()" function starting on page 128. Using this call back function, the USER can have certain functions executed (e.g., address an LED when a slave has failed). AMPRO-DPM is not aware of the effect of this function. Also applicable here: Functions not required by the USER must still be executed with at least a "return" command (i.e., dummy function).
For the individual function calls, see the sequence charts of the different functions.

5.1.3.3 DPM-DATR (AMPRO-DPM Data Transfer)

DPM-DATR is responsible for correct transmission of the data between USER and AMPRO2. In SLCB, the user must transfer one pointer to the input, output and diagnostic data area for each slave. AMPRO-DPM (i.e., the PROFIBUS ASIC or the processor on which AMPRO-DPM is being used) must be able to access these areas with normal memory commands or DMA transfers without any additional information since the current process data of DPM-DATR are directly transferred to these areas or read from these areas.
Depending on the requirements, the transmission of data can performed with various consistency assurance procedures. For further information, see the section on consistency assurance (page 70 ff.).

5.1.4 Consistency Assurance

Possible process data flows are divided into three groups for consistency assurance (i.e., inputs, outputs and diagnoses). For the inputs and outputs, one of the three possible types of consistency can be selected (i.e., "without", "short" or "long"). The selection is made by making an entry in the Slave_User_Data field of the slave parameter record. See the section on parameterization starting on page 73. In contrast, diagnostic data must always be transferred with "long" consistency.

At total of up to 244 bytes each of inputs, outputs and diagnoses can be transferred per slave. Depending on the type of consistency required, this maximum length may have to be reduced.

5.1.4.1 "Without" Consistency or Byte Consistency

The "without consistency" type of access actually means byte consistency. This type of access is divided into direct accesses and accesses in Buffered_Mode.

5.1.4.1.1 Direct Access

With direct access, AMPRO-DPM or the PROFIBUS ASIC takes all sending data directly from the output data area. All receiving data are entered directly in the input data area. Only the two pointers (entered in the SLCB) to the input and output areas are required. When an array of several bytes is involved, the data can be updated at every byte boundary (i.e., a contiguous information field may not be longer than one byte). This access technique ensures the fastest possible data update cycle and the transmission of large amounts of data up to 244 bytes. In addition, it is the simplest way to transfer data both for AMPRO-DPM and the USER. This type of data transmission can only be used for inputs under certain conditions, however. See the section on Buffered_Mode below.

5.1.4.1.2 Buffered_Mode

Due to several AMPRO2 characteristics, processing of the inputs without consistency is only possible up to a specified data length used by the PROFIBUS ASIC (ASPC2 STEP B: 58 bytes; ASPC2 STEP C: 122 bytes). When larger input data areas of a slave are involved, the inputs must be stored intermediately by AMPRO-DPM. AMPRO-DPM obtains this buffer from AMPRO2 memory management when "add_slave ()" is called based on the entries in the slave parameter record. The data may not be transferred to the process data area until receipt of the telegram is complete and correct. The processor must handle this copying procedure itself. The USER must keep the number of update cycles as low as possible so that the processor is not overloaded too frequently with telegrams (e.g., at 12 Mbaud). The procedure corresponds to "long" consistency. See below.
In Buffered_Mode, input data may not exceed 244 bytes per slave.

5.1.4.2 "Short" Consistency

Using appropriate entries in the AMPRO2 application blocks, the ASPC2 PROFIBUS ASIC can activate a hardware consistency mechanism before accessing the process data area. If the USER and the ASIC simultaneously access the process data area with consistency, the READY signal is withdrawn from one of the partners. The READY signal for access to the process data area is returned to the other partner immediately after the first partner concludes its access. This type of consistency permits the same simple and fast direct accesses as with byte consistency, even for contiguous information fields of more than 8 bits. In addition to READY withdrawal, the ASIC permits disabling with HOLD signals. For more detailed information, see the documentation of AMPRO2.
Basic prerequisite: The hardware must be equipped with suitable consistency control logic. The duration of the READY delay or the HOLD disable presents another problem. Since one of the partners is completely stopped during this time, it must be ensured that the amount of data transferred with "short" consistency does not become too large. When the USER host accesses the process data area for too long a period of time, this causes errors in telegram processing particularly when high baud rates are used. The maximum duration of such an access must be calculated empirically by the USER based on the particular hardware environment. This must usually be calculated for each baud rate separately. The maximum amount of data which can be transferred with "short" consistency (i.e., the *consistency limit*) is determined from the maximum duration. For ASPC2 STEP B, the consistency limit may not exceed 58 bytes in each direction. For ASPC2 STEP C, the limit is 122 bytes.

5.1.4.3 "Long" Consistency

When greater amounts of data are to be transferred than are possible with "short" consistency, "long" consistency must be set. "Long" consistency uses Buffered_Mode (i.e., when "add_slave ()" is called, AMPRO-DPM occupies several data buffers of AMPRO2 memory management. During cyclic data communication, AMPRO2 executes the AMPRO-DPM jobs with a set of these buffers. At certain points in time, these buffers are disabled by AMPRO-DPM and replaced with new ones. Transfer of the data to the process data area uses a consistency control which can be implemented on both hardware and software (e.g., semaphores). The USER can specify the type of lock. It only has to be set before the data are copied and then reset again. AMPRO-DPM uses a CBF to initiate the start of the copying procedure.
When the consistency lock is activated, one condition must be fulfilled, however. Activation may not cause a delay. For example, when the USER must copy data in a shared memory area with a CBF call of AMPRO-DPM, the user determines that the other party is accessing the shared memory area while the consistency lock is activated. In this case, the USER may not wait "inside" the CBF for the other party to finish the copying procedure which was just started. Instead, the USER must intermediately store the data to be copied somewhere else and conclude the CBF immediately. The USER cannot start copying the data to the required area until he has received access to the shared memory later.
The DP standard states that length and characteristics of the input and output data of a slave must be specified with configuration identifiers. One of these identifiers may describe a data area of up to 128 bytes in length. This also permits consistent areas with up to 128 bytes. When "long" consistency is used, slave data are processed by slave and not identifier. This permits use of long consistency data lengths of up to 244 bytes for the individual consistent areas.

5.1.5 Prerequisites for Use of AMPRO-DPM

5.1.5.1 Programming Language

AMPRO-DPM is written in programming language **C**. Since only **ANSI-C** elements were used, the call interface for AMPRO-DPM is the same for all **ANSI-C**-compatible **C** or **C++** compilers.
If the USER system is to be written in another programming language, the USER must adhere to the ANSI-C conventions when calling AMPRO-DPM functions. Since all global headers must also be rewritten in the new language, we recommend using an ANSI-C-type language for the USER system too.

5.1.5.2 Layer-2 Interface "AMPRO2"

The AMPRO2 layer-2 firmware must be used as the interface to layer 2 (part 1 of PROFIBUS standard). AMPRO2 is built directly on the PROFIBUS ASIC used and must be adapted to the particular ASIC type.
AMPRO2 must have been initialized before the first AMPRO-DPM call of a USER instance. The following parameters must be set, among others.

- "DP" operating mode
- Times in accordance with the required bus parameter record

The AMPRO-DPM software cannot check these parameters for plausibility. COM ET 200 for Windows (starting with V 1.0) is a suitable parameterization tool since it offers an easy-to-use environment and also handles the extensive calculations of the bus parameters suitable for the configured ET 200/DP system. In addition, AMPRO2 memory management must be initialized by the USER so that it can make both the layer-2 and layer-4 application blocks available. See section on resource requirements starting on page 75.
AMPRO-DPM uses only some of the functions and services provided by AMPRO2. These are listed in the following tables.

AMPRO2 functions:

- l2_mem_alloc_apb1 ()
- l2_mem_alloc_apb2 ()
- l2_mem_alloc_db1 ()
- l2_mem_alloc_db2 ()
- l2_mem_alloc_db3 ()
- l2_mem_alloc_db4 ()
- l2_mem_free_apb ()
- l2_mem_free_db ()
- l2_req ()

AMPRO2 service primitives:

- *Opcodes*
  - L2_FMA_REQUEST
  - L2_MAC_REP_CONFIRM_WITHDRAW
  - L2_MAC_REP_REQUEST
  - L2_MAC_REQUEST_HIGH
  - L2_REP_REQUEST

- *Opcode modifiers*
  Only two of these entries are required. The USER can specify in DPMIB which of the three definitions are to be used. See page 88 ff.
  - L2_L4_HLEN_FIXED
  - L2_L4_HLEN_VAR1
  - L2_L4_HLEN_VAR2

AMPRO2 services

- *Service codes*
  - L2_CLEAR_MODE_ACTIVATE
  - L2_CLEAR_MODE_DEACTIVATE**Fehler! Textmarke nicht definiert.**
  - L2_MAC_REP_EXCHANGE_REQ
  - L2_MAC_REP_EXCHANGE_RESP
  - L2_MAC_REP_EXCHANGE_REQ_RESP
  - L2_MAC_REP_EXCHANGE_REQ_SY
  - L2_MAC_REP_EXCHANGE_RESP_SY
  - L2_MAC_REP_EXCHANGE_REQ_RESP_SY
  - L2_MAC_REP_WITHDRAW
  - L2_MAC_REQ_LOCK
  - L2_MAC_REQ_UNLOCK
  - L2_SDN_HIGH
  - L2_SRD_HIGH
  - L2_SRD_LOW

- *Service code modifiers*
  - L2_RDCONS
  - L2_WRCONS

5.1.5.3 Parameterization

To process the slave, AMPRO-DPM requires from the USER a complete slave data record corresponding to the parameter module description. AMPRO-DPM expects this information in exactly this format. The USER transfers the slave data record by entering a suitable pointer in the SLCB. See section on the SLCB starting on page 82 and the "add_slave ()" function starting on page 96.
As long as the slave remains activated, AMPRO-DPM must be able to access this record at all times without any other mechanisms. This must be possible without any other mechanisms. The PROFIBUS ASIC used by AMPRO2 must also be able to access certain parts of this record (see table) directly. The following table shows which areas of a slave parameter record are used by which components.

Components of the Slave Parameter Record		Access by	
Name	**Designator**	**AMPRO-DPM**	**PROFIBUS ASIC**
S7 CPU header	-	No	No
COM-related header	-	Yes or no	No
General slave data	Slave_Para_Data	Yes	No
Parameterization data	Prm_Data	Yes	Yes
Configuration data	Cfg_Data	Yes	Yes
Address table	Add_Tab	No	No
Slave user data	Slave_User_Data	Yes	No
STS configuration data	STS_Cfg_Data	No (up to now)	No (up to now)
S7 substitute configuration data	S7_Ersatz_Cfg_Data	No (up to now)	No (up to now)
COM private data	-	No	No

The USER can transfer the parameter record to AMPRO-DPM as one complete record or in individual components. When the complete parameter record is transferred, AMPRO-DPM must determine the location of the individual components based on the required *COM-related headers*. When the parameter record is transferred in individual components, the USER transfers to AMPRO-DPM one pointer for each partial component. AMPRO-DPM can then disregard the COM header. See section on SLCB starting on page 82.
Since the areas *STS configuration data* and *S7 substitute configuration data* have not yet been defined, they have not been used either. These data blocks may find use in the future with later versions of AMPRO-DPM after appropriate specifications have been made for these areas in the parameter module description. Special notification will be provided.
The USER can use the master-related parameters also defined in the parameter module description (e.g., bus parameter record, master or host parameters, and so on) in any desired format since they must be taken by the USER from the parameter record for specific functions and entered in the AMPRO-DPM job blocks, and are not transferred directly to AMPRO-DPM.

We recommend using a suitable parameterization tool (e.g., COM ET 200 for Windows starting with version V 1.0) to generate these parameter records.
Although several entries are required as words in the parameter module description, the sequence of entry of the bytes within a word depends on the parameterization tool which was used to generate the parameter record. For example, COM ET 200 generates all word entries in Big-Endian format (see page 62) when exporting a parameter file to a memory card or as a binary file. However, Little-Endian format is used for exporting as an S7 data block. Macros which convert the word entries to the required format are used when processing the parameter record so that AMPRO-DPM can react to these differences. These macros are controlled by a compiler switch which must be entered in the "COMM_DEV\DEV_DEF.H" file: DPM_MODULE_FORMAT_xxxx_ENDIAN. In this definition, the string ..._xxxx_... must be replaced by the format of the parameter record (i.e., with ..._HIGH_... for High-Endian format or with ..._LITTLE_... for Little-Endian format). For more information on the "COMM_DEV\DEV_DEF.H" file, see part 3 of these specifications.

5.1.5.4 Consistency Monitoring

When the transmission of consistent data is required on the USER side, the USER must provide a suitable means of monitoring (hardware/software) the consistency requirements.
This operating mode is enabled on AMPRO-DPM when the parameter record is transferred. When the USER is unable to guarantee consistency, the USER must ensure that this parameter record does not contain a module which requires consistent processing. See also the section on consistency assurance starting on page 70.

5.1.5.5 Resource Requirements

Caution: All specifications in this section may change in the course of development or due to new functions or requirements.

5.1.5.5.1 Operating System

Both AMPRO2 and AMPRO-DPM are totally independent of the operating system. Use of an operating system depends on how many processors the USER wants to use in addition to AMPRO2 and AMPRO-DPM and the scope of these processes. In addition, the timer (see page 78) or stack and error handling may require an operating system if there is no other way to implement these functions. An operating system can be omitted when the required functions can be provided by other means and the scope of the USER programs is minimized.
The USER can also convert the CBF interface of AMPRO-DPM to a mailbox interface when necessary.

5.1.5.5.2 Memory

If not otherwise specified, all values given here for required memory refer to AMPRO-DPM only. Additional code and data memory is required for firmware packages such as AMPRO2, the operating system and the USER area.

AMPRO-DPM and the user require memory resources. Some of the resources for AMPRO-DPM must be provided by the USER with the AMPRO2's own memory management. This memory management is primarily block-oriented and can provide up to six different types of data blocks. This relieves the USER of managing memory during operation.
Before startup of AMPRO-DPM, the USER must initialize AMPRO2 memory management in addition to initializing the AMPRO2 layer-2 interface. This is performed with the "l2_mem_init_... ()" function whereby each call stands for a certain data area. The USER transfers the functions to a sufficiently dimensioned memory area which can hold all blocks of the type selected. Additional conditions apply to the location of the areas for application blocks. To put it simply, this area must be located within a 64-kbyte segment together with the SCB (i.e., System Control Block) of the ASIC. See also the *cb_segment_start_ptr* entry in DPMIB starting on page 88. In contrast, the AMPRO2 data blocks for the input and output area are only required for Buffered_Mode (see page 71 ff.) for slaves with "long" consistency or for large amounts of input data. They are allocated automatically by AMPRO-DPM when necessary.

The following is an example of total memory requirements for version V 1.0 for the IM 308-C.

Code	Total:	118	kB (approx.)
	Of this total, AMPRO2:	17	kB (approx.)
	Of this total, AMPRO-DPM:	37	kB (approx.)
Data	Data blocks:	128	kB (approx.)
	Application blocks:	23	kB (approx.)
	Total internal processing:	25.7	kB (approx.)
	Of this total, AMPRO2:	0.5	kB (approx.)
	Of this total, AMPRO-DPM:	0.5	kB (approx.)

However, each time new modules are added, conditions and environment must be examined again by the USER and the memory requirements of the new module calculated again.

5.1.5.5.2.1 Code

Since the amount of code depends on a number of factors (e.g., the compiler used to generate the code, the type and organization of the hardware and similar), the exact amount cannot be specified here. It is possible, however, to limit the total functionality of the firmware and reduce the amount of code. Generation of such subsets may affect the following areas.

- ☞ Without Siemens DP slaves
- ☞ Without processing using "long" consistency
- ☞ Without slave deactivation

These subsets are not yet contained in the code. They must be supplied based on USER requirements.

5.1.5.5.2.2 Data

AMPRO2-L2 application blocks (L2-APB):

Size: 24 bytes

The size depends on the memory model used. The specified value applies to the "SMALL" model. Use of models which require the FAR pointer for data increases memory requirements by at least 6 bytes.

Number (global): 14 APBs

- 8 "dummy_apb" APBs for the "set_slave_mode ()" function
- 1 "exchange_apb" APB, 1 "withdraw_apb" APB and 1 "lock_apb" APB for the "set_master_mode ()" function
- 1 "mark_apb" APB for the "mark_cycle ()" function
- 1 "ssla_apb" APB for the "set_slave_address ()" function
- 1 "bus_accessible" APB for the "bus_accessible ()" CBF

Number per slave: Max. of 7 APBs

- 1 "diag" APB for slave states DIAG1 and DIAG2
- 1 "cfg" APB for slave state CFG and the Read_Input service
- 1 "data" APB for slave state DATA and the Read_Output service
- 1 "withdraw_repeat" APB for the Withdraw_Repeat service
- 1 "change_i_buffer" APB for data communication (only required for slaves in Buffered_Mode or slaves with "long" consistency)
- 1 "change_o_buffer" APB for data communication (only required for slaves in Buffered_Mode or slaves with "long" consistency)
- 1 "clear" APB for the Clear_Data service (only required for Siemens DP slaves)

AMPRO2-L4 application blocks (L4-APB), type 1:

Size: 26 bytes (2 bytes more than the L2-APB)

This size also depends on the memory model used. The specified value applies to the "SMALL" model. Use of models which require the FAR pointer for data increases memory requirements by at least 6 bytes.

Number (global): 10 APBs

- 8 "ssm_apb" APBs for the "set_slave_mode ()" function
- 1 "smm_apb" APB for the Global_Control service, OPERATE and CLEAR functions, and for cycle monitoring
- 1 "tex_apb" APB for the "timer_expired ()" function

Number per slave: None

AMPRO2-L4 application blocks (L4-APB), type 2:

Size: 28 bytes (4 bytes more than the L2-APB)

This size also depends on the memory model used. The specified value applies to the "SMALL" model. Use of models which require the FAR pointer for data increases memory requirements by at least 6 bytes.

Number (global): None

Number per slave: 1 APB

- 1 "prm" APB for the PRM and PRM_UNLOCK slave states

AMPRO2 data blocks (L2-DB):

Size: Up to 244 bytes

Number (global): Max. of 3 DBs

- 1 DB with 244 bytes for the Global_Control service, OPERATE and CLEAR functions, and for "long" consistency or Buffered_Mode
- 1 DB with 244 bytes for the Set_Slave_Address service
- 1 DB with 6 bytes as scratchpad buffer

Number per slave: Max. of 9 DBs

- 3 DBs for input data (only for slaves with inputs which must be operated in Buffered_Mode)
- 3 DBs for output data (only for slaves with outputs which must be operated in Buffered_Mode)
- 2 DBs for diagnostic data. (The second DB is only required for DP Siemens slaves. Standard DP slaves require only one DB.)
- 1 DB for parameterization data (only for DP Siemens slaves with the "SPM" PROFIBUS ASIC)
- 1 DB for communication data (only for standard DP slaves in "Shared_IO" mode)

To save memory space, the sizes of the individual blocks set up by the USER should be as close as possible to the lengths required by the slaves. AMPRO2 memory management can handle up to four different block sizes for data blocks. One of the blocks must always have the maximum length of 244 bytes. The size of the other blocks should be selected so that as little unused space as possible is created in the

blocks for a parameterization. AMPRO-DPM evaluates the blocks sizes selected by the USER. See the *db_len_...* entries in DPMIB starting on page 88.

Other data areas:

Internal data: All internal data of AMPRO-DPM are located in the private portion of the DPM structure. The total size of the structure including the portion accessible by the USER is approx. 0.7 kbytes.

SLCBs: The USER must provide an SLCB for each slave. These have a minimum size of approx. 128 bytes. Since station addresses from 0 to 125 are permitted for the slaves, AMPRO-DPM can manage up to 125 slaves (one address is reserved for the master). When all 125 slaves are used, the SLCB memory address area has a minimum total size of approx. 15.7 kbytes.

This size is dependent on the memory model used and the distance of the components from each other. The specified value applies to code and data pointer with a size of NEAR (2 bytes). The SLCB contains approx. 31 different pointers. For models which use the 4-byte pointers for internal and external data and for the code (i.e., size: FAR), the size of the SLCB increases by 62 bytes to 190 bytes, thus giving the total SLCB memory area a size of approx. 23.2 kbytes.

Process data: The USER must also provide memory areas for the storage of process data. Three areas with a length of up to 244 bytes each are required for each slave. The exact length is dependent on the size of the data areas required by the slave.

- 1 area for input data
- 1 area for output data
- 1 area for diagnostic data

In addition to these areas, AMPRO-DPM requires a 48-byte area in the process data area for master diagnostics. See also the entry *master_diag_ptr* in DPMIB starting on page 88.

Job blocks: Additional job blocks for calling the AMPRO-DPM functions are also required. These blocks are only required while the particular function is being executed and can be used for other purposes later. To put it simply, all these blocks must be located within a 64-kB segment. See entry *cb_segment_start_ptr* in DPMIB starting on page 88. The block sizes are listed below.

- DPMIB: Approx. 66 bytes
- SMMCB: Approx. 20 bytes
- SSMCB: Approx. 22 bytes (required up to 8 times; see page 107 ff.)
- SSLACB: Approx. 30 bytes
- MARKCB: Approx. 20 bytes

Also here, the stated sizes depend on the pointer sizes used. See also the remark for SLCBs. Since each block must have at least two pointers, exact specification of the individual block sizes is impossible.

5.1.5.5.3 Timer

A timer must be provided by the USER for the various timer intervals required by the PROFIBUS-DP standard (e.g., DX_Control_Intervall and so on). See the "timer_expired ()" function starting on page 124.

5.1.5.5.4 Interrupt Handler

The following interrupt handlers are used.

- AMPRO2 interrupt handler of the USER
- Timer interrupt
- Handler for data consistency (when required)

When SPM slaves are used, interrupt reaction times must be shorter than 10 msec. No special requirements exist for other types of slaves. However, the effect of all delay times on all PROFIBUS time parameters must be allowed for.

5.1.5.5.5 Disable Times

AMPRO-DPM does not trigger disable times for itself, but disable times do occur while AMPRO2 is being executed.
In certain situations, AMPRO-DPM may not be interrupted by an AMPRO2 interrupt. This is locked by AMPRO-DPM. The interrupt locking functions must be supplied to AMPRO-DPM by the USER. In the simplest cases, these functions only consist of clearing or setting the interrupt enable flag. They are always used in pairs by AMPRO2 and AMPRO-DPM. However, the USER should ensure (e.g., with one of counters used by both functions) that a multiple call of the "interrupt disable" function is not actually enabled again until after the same number of calls of the "interrupt enable" function.

Run times of the jobs are dependent on the specific system. The various call of the "state_report ()" function can be used to determine the individual AMPRO-DPM run times for the changes in state. See description of this function starting on page 128. Several measurements should be performed for each change in state to eliminate isolated delays caused by rare interrupts. In addition, this procedure can be used to determine the run time of the USER software at the same time.

5.2 User Interface

5.2.1 Call Structures

5.2.1.1 Call Structure for AMPRO-DPM

AMPRO-DPM provides the USER with an Includefile in which all structures relevant to the USER are defined. One of theses call structures is the call structure for the AMPRO-DPM functions. This structure can be used by the USER to utilize most of the AMPRO-DPM functions. The transfer parameters of the functions are usually data blocks containing additional information on the particular service. To improve the organization of the programs, each of the data block structures and the pointers to these data block structures has been provided with a separate definition. The files containing the designators are located in the "\COMMON" directory of the transferred sources. These definitions are listed below as applicable to the structure described here.

```
#define    DPM        struct dpm_def                        /* File DPM_COMM.H */

#define    SLCB       struct slcb_def                       /* File DPM_COMM.H */
#define    DPMIB      struct dpmib_def
#define    SMMCB      struct smmcb_def
#define    SSMCB      struct ssmcb_def
#define    SSLACB     struct sslacb_def
#define    MARKCB     struct markcb_def

#define    ERRCB      struct errcb_def               /* File: DP_ERROR.H */


#define    DPM_PTR    DPM        DPM_INT_DATA_ATTR    *    /* File: DPM_COMM.H */

#define    SLCB_PTR   SLCB       DPM_IFA_DATA_ATTR    *    /* File: DPM_COMM.H */
#define    DPMIB_PTR  DPMIB      DPM_IFA_DATA_ATTR    *
#define    SMMCB_PTR  SMMCB      DPM_IFA_DATA_ATTR    *
#define    SSMCB_PTR  SSMCB      DPM_IFA_DATA_ATTR    *
#define    SSLACB_PTR SSLACB     DPM_IFA_DATA_ATTR    *
#define    MARKCB_PTR MARKCB     DPM_IFA_DATA_ATTR    *

#define    ERRCB_PTR  ERRCB      DPM_IFA_DATA_ATTR    *    /* File: DP_ERROR.H */
```

This results in the following structure definition.

```
DPM  /* File: DPM_COMM.H */
{
     /* public part */

     Unsigned16      DPM_IFA_FUNC_ATTR    (*close)             (Unsigned8);
     Unsigned16      DPM_IFA_FUNC_ATTR    (*init)              (DPMIB_PTR);
     Unsigned16      DPM_IFA_FUNC_ATTR    (*add_slave)         (SLCB_PTR);
     Unsigned16      DPM_IFA_FUNC_ATTR    (*withdraw_slave)    (SLCB_PTR);
     Unsigned16      DPM_IFA_FUNC_ATTR    (*suspend_slave)     (SLCB_PTR);
     Unsigned16      DPM_IFA_FUNC_ATTR    (*restart_slave)     (SLCB_PTR);
     Unsigned16      DPM_IFA_FUNC_ATTR    (*set_master_mode)   (SMMCB_PTR);
     Unsigned16      DPM_IFA_FUNC_ATTR    (*set_slave_mode)    (SSMCB_PTR);
     Unsigned16      DPM_IFA_FUNC_ATTR    (*input_update)      (SLCB_PTR);
     Unsigned16      DPM_IFA_FUNC_ATTR    (*output_update)     (SLCB_PTR);
     Unsigned16      DPM_IFA_FUNC_ATTR    (*consistency_update)(void);
     Unsigned16      DPM_IFA_FUNC_ATTR    (*timer_expired)     (void);
     Unsigned16      DPM_IFA_FUNC_ATTR    (*mark_cycle)        (MARKCB_PTR);
     Unsigned16      DPM_IFA_FUNC_ATTR    (*set_slave_address) (SSLACB_PTR);

     void            SYS_CBD_FUNC_ATTR    (*dpm_l2_cb_server)  (L2_APB_PTR);


     /* private part */
```

```
      ...
      ...  /* This part does not apply to the USER ! */
      ...
};
```

To be able to use the AMPRO-DPM functions, the USER must first call the "dpm_open ()" function (similar to a constructor). The return value of "dpm_open()" is the pointer to the AMPRO-DPM functions as shown in the above listed structure. See description of the "dpm_open ()" function starting on page 86. AMPRO-DPM sets this pointer to the beginning of the structure so that the USER can use this pointer to utilize the remaining AMPRO-DPM functions.
The return value of all AMPRO-DPM functions is a status word which provides the USER with an identifier indicating the status of the current function. See page 154 ff. and the description of the functions.

5.2.1.2 Call Structure for the USER

Some functions (i.e., the CBFs) must be provided by the USER for AMPRO-DPM. These functions must comply with the following declarations.

```
void DPM_CALL_BACK_FUNC_ATTR   (*init_done)              (Unsigned16);
void DPM_CALL_BACK_FUNC_ATTR   (*state_report)           (SLCB_PTR);
void DPM_CALL_BACK_FUNC_ATTR   (*withdraw_slave_done)    (SLCB_PTR, Unsigned16);
void DPM_CALL_BACK_FUNC_ATTR   (*set_master_mode_done)   (SMMCB_PTR, Unsigned16);
void DPM_CALL_BACK_FUNC_ATTR   (*set_slave_mode_done)    (SSMCB_PTR, Unsigned16);
void DPM_CALL_BACK_FUNC_ATTR   (*set_slave_address_done) (SSLACB_PTR, Unsigned16);
void DPM_CALL_BACK_FUNC_ATTR   (*mark_cycle_done)        (MARKCB_PTR, Unsigned16);
void DPM_CALL_BACK_FUNC_ATTR   (*consistency_update_done)(Unsigned16);

void DPM_CALL_BACK_FUNC_ATTR   (*set_timer)              (Unsigned32);
void DPM_CALL_BACK_FUNC_ATTR   (*bus_accessible)         (Boolean);

void DPM_CALL_BACK_FUNC_ATTR   (*error)                  (ERRCB);

void DPM_CALL_BACK_FUNC_ATTR   (*write_inp_data_to_pda)
                                    (SLCB_PTR, Unsigned8 L2_DATA_ATTR **);
void DPM_CALL_BACK_FUNC_ATTR   (*read_outp_data_from_pda)
                                    (SLCB_PTR, Unsigned8 L2_DATA_ATTR **);
void DPM_CALL_BACK_FUNC_ATTR   (*write_diag_data_to_pda)
                                    (SLCB_PTR, Unsigned8 L2_DATA_ATTR **);
void DPM_CALL_BACK_FUNC_ATTR   (*copy_s7_get_cfg_data)
                                    (SLCB_PTR, Unsigned8 L2_DATA_ATTR **, Unsigned8);
void DPM_CALL_BACK_FUNC_ATTR   (*clear_cons_input_data)
                                    (SLCB_PTR, Unsigned8 L2_DATA_ATTR **, Unsigned8);

void DPM_CALL_BACK_FUNC_ATTR   (*input_update_done)      (SLCB_PTR);
void DPM_CALL_BACK_FUNC_ATTR   (*output_update_done)     (SLCB_PTR);

void DPM_CALL_BACK_FUNC_ATTR   (*asic_int_disable)       (void);
void DPM_CALL_BACK_FUNC_ATTR   (*asic_int_enable)        (void);
```

None of the USER CBFs permit return values to AMPRO-DPM. When the USER is unable to execute a function within a CBF and this error also affects communication with AMPRO-DPM, the USER must set an appropriate USER flag, conclude the CBF, and execute his reaction to the flag setting with a new command to AMPRO-DPM if necessary.

Since all AMPRO-DPM functions are not reentrant (see section on communication model starting on page 61), this procedure must also be used when the USER wants to call an additional AMPRO-DPM function in reaction to an AMPRO-DPM CBF. A USER call directly from the CBF is not permitted.

Most USER CBFs must be able to be called within the USER call. For more details, see description of the CBFs starting on page 127.

5.2.2 Slave Control Block (SLCB)

The user provides one job data block (i.e., the Slave Control Block (SLCB)) per slave for the slave-related functions. The easiest way for the USER to manage the SLCBs is in an array which the USER sets up permanently for the maximum possible number of slaves. A possible definition is shown below.

```
SLCB slcb[MAX_ANZ_SLAVES];
```

Before the block can be used for the first job, the user must enter a pointer to the slave parameter record of the current slave in the SLCB (among others). AMPRO-DPM reads all internally required parameters from this area, and enters these and other internal data records in the private area of the SLCB which the USER cannot access. For this reason, each of the slave parameter records must be located in a memory area which can be accessed by AMPRO-DPM. In addition, this memory area must be able to be accessed **without** using any special access mechanisms. The USER must also provide pointers to all areas applicable to data maintenance (e.g., pointers to the areas for input, output and diagnostic data, and so on).

The following tables show the three-part layout of the SLCB for each slave. The entries can be used as desired before the SLCB is transferred to AMPRO-DPM for the first time. Not until the "add_slave ()" call is executed (see page 96 ff.) do the following access restrictions take effect.

5.2.2.1 Header

The header shown below precedes all job blocks still to be described below. The header is used for identification of the USER, the job and for internal AMPRO-DPM chaining of several blocks.

Type	Designation	Direction of Access			
		USER		DPM	
		L	S	L	S
DPM_IFA_DATA_ATTR *	next_blk_ptr			X	X
DPM_IFA_DATA_ATTR *	prev_blk_ptr			X	X
Unsigned8	opcode				
Unsigned8	subsystem	X	X	X	
L2_TYPE_ID_PTR	id_ptr	X	X		

next_blk_ptr:
prev_blk_ptr:

Internal AMPR-DPM chain control pointer to the next or previous SLCB. After transferring the SLCB to AMPRO-DPM for the first time, the USER may never access these pointers again. See "add_slave ()" function starting on page 96.

opcode:

The currently requested function identifier is usually transferred here. The byte is not used here since it is not applicable to the SLCB.

subsystem:

Based on this variable, the USER identifies himself to AMPRO-DPM. It contains the *dpm_handle* which is used to describe the "dpm_open ()" function in more detail. See page 86 ff.

id_ptr:

This pointer can be used as desired by the USER. It can be used as an integer value instead of a pointer. For example, it can be used by the USER to recognize the SLCB again within the USER's own management.

5.2.2.2 General Portion

This area of the SLCB is processed by both USER and AMPRO-DPM. Several entries must be provided by the USER while AMPRO-DPM informs the USER of others. During operation, the USER can access current information from this portion after receiving access rights to the SLCB.

Type	Designation	Direction of Access			
		USER		DPM	
		L	S	L	S
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to the "state_report ()" CBF	X	X	X	
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to the "withdraw_slave_done ()" CBF	X	X	X	
Unsigned8	ts	X	X	X	
Unsigned8	type	X	X	X	
Unsigned8	last_state	X		X	X
Unsigned8	actual_state	X		X	X
DPM_MOD_DATA_ATTR *	slave_para_set_ptr (pointer to the MOD_DATA_HEADER structure)	X	X	X	
DPM_MOD_DATA_ATTR *	para_data_ptr (pointer to the SLAVE_PARA_DATA structure)	X	X	X	
DPM_MOD_DATA_ATTR *	prm_data_ptr (pointer to the SLAVE_PRM_DATA structure)	X	X	X	
DPM_MOD_DATA_ATTR *	cfg_data_ptr (pointer to the SLAVE_CFG_DATA structure)	X	X	X	
DPM_MOD_DATA_ATTR *	user_data_ptr (pointer to the SLAVE_USER_DATA structure)	X	X	X	
DPM_MOD_DATA_ATTR *	sts_cfg_data_ptr (pointer to the SLAVE_STS_CFG_DATA structure)	X	X	X	
DPM_MOD_DATA_ATTR *	s7_cfg_data_ptr (pointer to the SLAVE_S7_CFG_DATA structure)	X	X	X	
DPM_PROC_DATA_ATTR *	input_data_ptr (pointer to an Unsigned8 array)	X	X	X	
Unsigned8	input_data_len	X	X	X	
Unsigned8	input_data_db_no	X		X	X
Unsigned8	Reserved for later developments				
DPM_PROC_DATA_ATTR *	output_data_ptr (pointer to an Unsigned8 array)	X	X	X	
Unsigned8	output_data_len	X	X	X	
Unsigned8	output_data_db_no	X		X	X
Unsigned8	special_functions	X	X	X	
DPM_PROC_DATA_ATTR *	diag_data_ptr (pointer to the SLAVE_DIAG structure)	X	X	X	
DPM_PROC_DATA_ATTR *	act_diag_data_len_ptr (pointer to an Unsigned16 value)	X	X	X	
Unsigned8	act_diag_data_len	X		X	X
Boolean	new_diag_data	X		X	X
Unsigned8	max_s7_cfg_data_len	X	X	X	
Unsigned8	max_s7_cfg_data_db_no	X		X	X

Pointer to the "state_report ()" CBF:
Pointer to the"withdraw_slave_done ()" CBF:

Here, the USER can enter for each slave the function which is to be called as a call back function at a certain time. See also the section on USER CBFs starting on page 127. The USER decides whether to provide one function for each slave or use one function for several slaves.

ts:

Slave station number from 0_D to 125_D. Station number 126_D is reserved as the default address for special applications. *ts* must be entered by the USER.

type:

The appropriate slave type must also be entered by the USER. This corresponds to the declarations in the parameter model description. Several predefined definitions are permitted for *type*. These are summarized in the table of slave types on page 155.

last_state:
actual_state:

Current and last state of the slave. Possible entries include SL_DEACT, SL_DIAG1, SL_PRM, SL_CFG, SL_DIAG2, SL_DATA and so on. See the "state_report ()" function starting on page 128 and the table on page 155. These states must not be initialized by the USER. This is handled by the "add_slave ()" function. See page 96 ff.

slave_para_set_ptr:

para_data_ptr:
prm_data_ptr:
cfg_data_ptr:
user_data_ptr:
dst_cfg_data_ptr:
s7_cfg_data_ptr:

A complete slave parameter record must be prepared for each slave. See section on parameterization starting on page 73. Except for the station number and the slave type in the first partial parameter record (Slave_Para_Data), the entries in this parameter record are not checked for plausibility. It is up to the USER to ensure that the specifications of the parameter module description are adhered to for the slave parameter record. The best way to ensure this is to use COM ET 200 for Windows (starting with version V 1.0) to create the parameter records. In addition to the specifications from the parameter module description, the parameter record must fulfill other requirements which have already been described in the section mentioned above.
Two methods are available to the USER to transfer the slave parameter record to AMPRO-DPM. One method is to create a contiguous parameter record or take it from the data base. In the *slave_para_set_ptr* entry of the SLCB, the USER transfers a type MOD_DATA_HEADER_PTR pointer to the COM header of this record. AMPRO-DPM uses the COM header to evaluate the complete record, determine (if necessary) the component pointers, and enter these in the following pointers. The component pointers may not be read or write-accessed by the USER.
In certain situations, it is better to split up the parameter record and store the individual components at different locations instead of storing the complete parameter record in one place. AMPRO-DPM can then no longer use the COM header to determine the location of the individual components, and the USER must supply this information. To do this, the USER enters one pointer to each partial parameter record in the pointers *para_data_ptr*, *prm_data_ptr*, *cfg_data_ptr*, *user_data_ptr*, *dst_cfg_data_ptr* and *s7_cfg_data_ptr* of the SLCB. This makes the COM header irrelevant for AMPRO-DPM. To indicate this situation, the USER must enter the value ZERO in *slave_para_set_ptr*. AMPRO-DPM then uses the pointers entered in the structure without further modification.

input_data_ptr:

Pointer to input data (Unsigned8 array) in the process data area. When input data are available on the slave, AMPRO-DPM uses this area to store the input data of the slave in accordance with the receiving telegram. This pointer can be disregarded when the slave is not equipped with inputs.

input_data_len:

Length of the input data. When the slave is not equipped with inputs, 0_H must be entered here.

input_data_db_no:

When the inputs of the slave must be transferred with "long" consistency (see section on consistency assurance starting on page 70), AMPRO-DPM enters the number of the data block size required for these input data here during the "add_slave ()" function (see page 96). If the USER wants to transfer a new buffer to AMPRO-DPM during the call of the "write_inp_data_to_pda ()" CBF (see page 141 ff.), this value is helpful when selecting the appropriate DB allocation function of AMPRO2 memory management.
This entry can be disregarded when another type of consistency is defined for the slave. In this case, FF_H is entered here.

output_data_ptr:

Pointer to output data (Unsigned8 array) in the process data area. When output data are available on the slave, AMPRO-DPM reads the output data of the slave from this area and sends them. This pointer can be disregarded when the slave is not equipped with outputs.

output_data_len:

Length of the output data. When the slave is not equipped with outputs, 0_H must be entered here.

output_data_db_no:

When the outputs of the slave must be transferred with "long" consistency (see section on "long" consistency), AMPRO-DPM enters the number of the data block size required for these output data here during the "add_slave ()" function (see page 96). If the USER wants to transfer a new buffer to AMPRO-DPM during the call of the "read_outp_data_from_pda ()" CBF (see page 141 ff.), this value is helpful when selecting the appropriate DB allocation function of AMPRO2 memory management.
This entry can be disregarded when another type of consistency is defined for the slave. In this case, FF_H is entered here.

special_functions:

When the slave requires special handling in regard to AMPRO-DPM, the USER must announce this in special_functions. The entry is bit-coded.
Currently, special handling is only available for slave status S7_GET_CFG on BIT 0 (1 = activated; 0 = deactivated). To activate this special handling, the Special_Function entry must be assigned DPM_SL_SF_S7_GET_CFG for activation or DPM_SL_SF_NO_S7_GET_CFG for deactivation.

diag_data_ptr:

Type SLAVE_DIAG pointer to diagnostic data in the process data area. AMPRO-DPM stores the diagnostic data of the slave in this area.

act_diag_data_len_ptr:

Pointer to the diagnostic data length (Unsigned16) in the process data area. AMPRO-DPM stores the current length of the diagnostic data of the slave in this area in the form of a byte entry. This is also done for the subsequent SLCB entry. Even when this entry in the process data area is not required by the USER, the address of a dummy byte must still be specified by the USER.

diag_data_db_no:

During the "add_slave ()" function (see page 96), AMPRO-DPM enters the number of the data block size which is required for these diagnostic data here. If the USER wants to transfer a new buffer to AMPRO-DPM during the call of the "write_diag_data_to_pda ()" CBF (see page 142 ff.), this value is helpful when selecting the appropriate DB allocation function of AMPRO2 memory management.

act_diag_data_len:

Current length of the diagnostic data in the process data area

new_diag_data:

When a new diagnosis was generated during the last change in state of the slave, this variable contains the value DP_TRUE. See page 154. The USER can then evaluate the diagnostic data in the process data area up to the length specified in "act_diag_data_len ". When no diagnosis was generated during the last change in state of the slave, this variable contains the value DP_FALSE.

max_s7_cfg_data_len:

Length of the configuration data after startup of an S7_GET_CFG slave

max_s7_cfg_data_db_no:

When the current slave is a type S7_GET_CFG slave, AMPRO-DPM enters the number of the data block size required for these output data during the "add_slave ()" function. See page 96. If the USER wants to transfer a new buffer to AMPRO-DPM during the call of the "copy_s7_get_cfg_data ()" CBF (see page 141 ff.), this value is helpful when selecting the appropriate DB allocation function of AMPRO2 memory management.

5.2.2.3 Private Portion for AMPRO-DPM

This area is used by AMPRO-DPM to store values which are required internally. The USER may not read or write-access this area.

5.2.3 AMPRO-DPM Functions

AMPRO-DPM calls may not be called again during a call. The USER must ensure that one of his interrupt routines does not call a function which was already started by him during his normal program. AMPRO-DPM cannot check this for plausibility, howver.
When a CBF is called in addition to a USER job (e.g., during the "set_master_mode ()" function - see description starting on page 104, or the "set_master_mode_done ()" CBF - see description starting on page 138), the USER may also not call the same job again as long as this sequence has not yet been concluded. Exception: "set_slave_mode ()" function (see description starting on page 107). If attempted anyway, the USER receives an error message as return value since this USER error can be monitored by AMPRO-DPM.

*5.2.3.1 dpm_open (Unsigned8 DPM_IFA_DATA_ATTR *)*

5.2.3.1.1 Description

To be able to use the AMPRO-DPM functions, the USER must first call the "dpm_open ()" function (similar to a constructor; definition in source file "\COMMON\DPM_COMM.H"). This is the only AMPRO-DPM function

which the USER **cannot** start via the "DPM" AMRPO-DPM call structure (see page 80 ff.) since the USER does not receive the pointer to the AMPRO-DPM call structure until the "dpm_open ()" function.
The USER transfers the address of a byte variable to the function. AMPRO-DPM provides a permanently defined, global memory area for the AMPRO-DPM call structure itself. During this function, "dpm_open ()" specifies a handle which is stored in the byte variable addressed by the pointer. This handle must be entered for USER identification in the job blocks (in the *subsystem* component) for all subsequent AMPRO-DPM jobs. This handle has no other meaning for the USER.
The return value of the function is the already stated pointer to the AMPRO-DPM call structure. This structure does not yet contain all function pointers since the USER must call the "init ()" function (see page 88 ff.) before AMPRO-DPM can be used again. To make this procedure fail safe, the AMPRO-DPM call structure contains (after "dpm_open ()") only the pointers to the "init ()" and "close ()" functions (see page 126 ff.) and the pointer to the "dpm_l2_cb_server ()" interrupt distributor (see page 127 ff.). The other pointers still have the value ZERO. They are entered before the "init_done ()" CBF is called. See page 128 ff.

5.2.3.1.2 Call

Sample call:

The following sample program illustrates the access procedure. After "dpm_open ()", the "add_slave ()" function is called for slave x. The variables and function definitions specified here also apply to the sample programs of other functions.

Function definitions from the "\COMMON\DPM_COMM.H" file:

```
extern     DPM_PTR    DPM_IFA_FUNC_ATTR    dpm_open
                                           (Unsigned8 DPM_IFA_DATA_ATTR *);
```

Required variables:

```
SLCB       DPM_IFA_DATA_ATTR    slcb[MAX_ANZ_SLAVES];
DPM_PTR                         dpm_ptr;
Unsigned8                       dpm_handle;
DPMIB                           dpmib;

Unsigned16                      status;
```

Function parts:

```
/* Call OPEN function */
dpm_ptr = dpm_open ((DPM_UNSIGNED8_PTR) &dpm_handle);

     ...

/* Complete DPMIB and call init */
dpmib.opcode    = DPM_INIT;
dpmib.subsystem = dpm_handle;
dpmib.id_ptr    = (L2_TYPE_ID_PTR) 0x0815;
     ...
     ...
     ...
status = dpm_ptr -> init ((DPMIB_PTR) &dpmib);

     ...

/* Complete SLCB and call add_slave */
slcb[x].subsystem    = dpm_handle;
slcb[x].id_ptr       = (L2_TYPE_ID_PTR) x;
     ...
     ...
     ...
status = dpm_ptr -> add_slave ((SLCB_PTR) &slcb[x]);
```

5.2.3.2 init (DPMIB_PTR)

5.2.3.2.1 Description

Before the "init ()" function can be called, AMPRO2 memory management must have been initialized and an AMPRO2 reset executed. In addition, the "init ()" function must be the first AMPRO-DPM function to be called after "dpm_open ()".
By calling the function, AMPRO-DPM initializes all internal data structures, reports to AMPRO2 with "l2_open ()", obtains the required memory blocks from AMPRO2 memory management, and fills in the internal portion of the DPM structure. AMPRO-DPM then fills in all already detected, global application and data blocks in accordance with the specifications of the particular services. After initialization, AMPRO-DPM attempts to send an initial telegram via the bus to check its functionality. Not until this is successfully accomplished does the USER receive the "init_done ()" CBF with the return message that the "init ()" function has been concluded.
The first telegram is sent immediately before conclusion of the "init ()" function, and the "init_done ()" CBF is usually not called until the end of "init ()". Particularly when high baud rates are used, telegram transmission may have already been concluded before AMPRO-DPM is able to conclude the "init ()" function. If this happens, the USER receives the "init_done ()" return message CBF while the "init ()" function is still running. This can happen in any of the functions still to be described. The USER cannot be absolutely sure that the functions will be processed in strict sequence.

This function may only be executed once. A new start is only permitted after the "close ()" function is called. See page 126 ff. A second call of "init ()" before "close ()" is rejected by AMPRO-DPM with an error message. The transfer parameter is a completed block of the DPMIB structure. The layout of this block is shown below.

5.2.3.2.2 DPM Init Block (DPMIB)

5.2.3.2.2.1 Header

Type	Designation	Direction of Access			
		USER		DPM	
		L	S	L	S
DPM_IFA_DATA_ATTR *	next_blk_ptr			X	X
DPM_IFA_DATA_ATTR *	prev_blk_ptr			X	X
Unsigned8	opcode (DPM_INIT)	X	X	X	
Unsigned8	subsystem (dpm_handle)	X	X	X	
L2_TYPE_ID_PTR	id_ptr (can be used by the USER as desired)	X	X		

The header is completed similar to the header description of the SLCB. See page 82 ff. The function identifier DPM_INIT (see page 156) must be entered here for *opcode* and the handle supplied by the "dpm_open ()" function must be entered for *subsystem* (see page 86 ff.).

5.2.3.2.2.2 General Portion

Type	Designation	Direction of Access			
		USER		DPM	
		L	S	L	S
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to the "asic_int_enable ()" CBF	X	X	X	
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to the "asic_int_disable ()" CBF	X	X	X	
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to the "set_timer ()" CBF	X	X	X	
DPM_CALL_BACK_ FUNC_ATTR *	*Reserved for later developments*				
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to the "bus_accessible ()" CBF	X	X	X	
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to the "init_done ()" CBF	X	X	X	
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to the "error ()" CBF	X	X	X	
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to the "write_inp_data_to_pda ()" CBF	X	X	X	
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to the "read_outp_data_from_pda ()" CBF	X	X	X	
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to the "consistency_update_done ()" CBF	X	X	X	
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to the "copy_s7_get_cfg_data ()" CBF	X	X	X	
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to the "write_diag_data_to_pda ()" CBF	X	X	X	
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to the "clear_cons_input_data ()" CBF	X	X	X	
DPM_PROC_DATA_ATTR *	master_diag_ptr (pointer to the MASTER_DIAG_DATA) structure)	X	X	X	
DPM_IFA_DATA_ATTR *	cb_segment_start_ptr	X	X	X	
Unsigned32	min_slave_interval	X	X	X	
Unsigned32	dx_control_interval	X	X	X	
Unsigned8	*Reserved for later developments*				
Unsigned8	ts	X	X	X	
Unsigned8	db_len_1	X	X	X	
Unsigned8	db_len_2	X	X	X	
Unsigned8	db_len_3	X	X	X	
Unsigned8	db_len_4	X	X	X	
Unsigned8	sync_mask	X	X	X	
Unsigned8	freeze_mask	X	X	X	
Unsigned8	l2_apb_func_no	X	X	X	
Unsigned8	*Reserved for later developments*				
Unsigned8	l4_apb_type1_func_no	X	X	X	
Unsigned8	l4_apb_type1_opcode_mask	X	X	X	
Unsigned8	l4_apb_type2_func_no	X	X	X	
Unsigned8	l4_apb_type2_opcode_mask	X	X	X	
Unsigned8	special_functions	X	X	X	
Unsigned8	asic_type	X	X	X	
Unsigned8	master_class	X	X	X	
Unsigned8	*Reserved for later developments*				

Pointer to the "asic_int_enable ()" CBF:
Pointer to the "asic_int_disable ()" CBF:

The USER must enter the routines for disabling and enabling the ASIC interrupt here. For a description of each CBF, see page 127 ff.

Pointer to the "set_timer ()" CBF:
theCBFPointer to the "bus_accessible ()" CBF:
Pointer to the "init_done ()" CBF:
Pointer to the "error ()" CBF
Pointer to the "write_inp_data_to_pda ()" CBF
Pointer to the "read_outp_data_from_pda ()" CBF
Pointer to the "write_diag_data_to_pda ()" CBF
Pointer to the "input_update_done ()" CBF
Pointer to the "output_update_done ()" CBF

These pointers must point to the individual CBF. For a description of each CBF, see page 127 ff.

master_diag_ptr:

AMPRO-DPM maintains a list with various master and slave-related information to provide the USER with a quick overview of the status of all AMPRO-DPM functions. The layout of this list is shown below.

Area	*Type*	*Meaning*	*Explanation*
System_Diagnostic (SD)	bit	Slave no. 0_D	Bit no. X = 0_B: Slave no. X has not reported diagnoses. Bit no. X = 1_B: Slave no. X has reported diagnoses.
	bit	Slave no. 1_D	
	bit	Slave no. 2_D	
	:	:	
	bit	Slave no. 124_D	
	bit	Slave no. 125_D	
	bit[2_D]	*Reserved*	-
Master_Status	Unsigned8	USIF_State	Master status: MA_STOP, MA_CLEAR or MA_OPERATE
	Unsigned8	Ident_Number_high	Hardware ident number high
	Unsigned8	Ident_Number_low	Hardware ident number low
	Unsigned8	MASTER_HW_Version	AMPRO-DPM hardware version
	Unsigned8	MASTER_FW_Version	AMPRO-DPM firmware version
	Unsigned8	USER_HW_Version	Hardware version of USER
	Unsigned8	USER_FW_Version	Firmware version of USER
	Unsigned8 [9_D]	*Reserved*	-
Data_Transfer_List (DTL)	bit	Slave no. 0_D	Bit no. X = 0_B: Slave no. X is not in DATA status. Bit no. X = 1_B: Slave no. X is in DATA status.
	bit	Slave no. 1_D	
	bit	Slave no. 2_D	
	:	:	
	bit	Slave no. 124_D	
	bit	Slave no. 125_D	
	bit[2_D]	*Reserved*	-

This **master diagnosis** (MASTER_DIAG_DATA structure; defined in the "\COMMON\DPM_COMM.H" file) contains the three master diagnostic information areas described in the PROFIBUS standard (i.e., "Master_Status", "Data_Transfer_List" (DTL) and "System_Diagnostic" (SD). In addition to the information for the USER, the USER can use this field to inform another master (e.g., a class-2 DP master) of his own hardware and firmware release status and his ident number.
Each of the three areas of the master diagnostic field which can be accessed by AMPRO-DPM via the transferred pointer has a length of 16_D bytes for a total length of exactly 48_D bytes.

Before the "init ()" function is called, the USER initializes the entries *Ident_Number*, *MASTER_HW_Version*, *USER_HW_Version* and *USER_FW_Version*. These entries are accepted by AMPRO-DPM without a plausibility check. The entries for *MASTER_HW_Version* and *USER_HW_Version* will usually be the same, however. The USER now calls the "init ()" function. AMPRO-DPM enters its own firmware version identifier in the *MASTER_FW_Version* field of the "Master_Status" area and the MA_STOP state in the *USIF_State* field.

The relevant bits of the "Data_Transfer_List" and "System_Diagnostic" areas are preset with 0_B. From this time on, the entire field is cyclically updated by AMPRO-DPM (i.e., after the "init ()" function is called, the USER may only read-access this field but never write-access it).

The "Data_Transfer_List" and "System_Diagnostic" areas are bit fields. One bit is assigned to each slave station. The two bit fields must be combined into 8 words each for generation of the firmware which is stored in Big-Endian (Intel™) format. See also section on the sequence of the bytes of a word on page 62. The following table shows the slave station numbers assigned to each bit.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word																
0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
7	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
8	-	-	125	124	123	122	121	120	119	118	117	116	115	114	113	112

A bit in the "Data_Transfer_List" area is always set when the corresponding slave has achieved or retained the DATA state at least once during the past three (approx.) data cycles. For more details on the slave states, see the "state_report ()" function. The list is cyclically cleared at intervals of "Dx_Control_Interval / 2" (see below) and set up again. The status before clearing takes place is always entered in the master diagnostic area. For more details, see the PROFIBUS-DP standard.

A bit is set for the first time in the "System_Diagnostic" area when the corresponding slave exits the DEACT state. This bit is reset when the slave changes to the DATA state. The bit is always set during processing when the slave reports diagnoses (i.e., state not DATA or DEACT). In other words, a single diagnostic telegram can also cause the bit to be set during the data cycle (status transition → DIAG2 → DATA). The bit is also set in STOPPED status.

cb_segment_start_ptr:

All job blocks for AMPRO-DPM must be located within a 64-kB segment. See also section on required memory resources starting on page 75. The location of this segment can be selected as desired by the USER except that the SCB (System Control Block) required for the PROFIBUS ASIC must be located in the same segment. In addition, the segment must always start at a real segment boundary (offset 0000_H) if the entire segment is to be used.
The *cb_segment_start_ptr* entry is used by the USER to transfer to AMPRO-DPM a pointer to the beginning of the segment. The job blocks can be distributed anywhere within the segment except that the beginning of the first block and the end of the last block may not extend outside the segment. The remaining areas of this segment can also be used as desired.
This special feature is due to the fact that AMPRO-DPM only has one 16-bit word at its disposal for correct allocation of an AMPRO2 confirmation. To be able to support all sizes of the memory model of the user despite this restriction, the 16-bit word is used as an offset to the *cb_segment_start_ptr* whose size can be selected as desired.

min_slave_interval:
dx_control_interval:

The entries required for the two timer parameters are described in detail by the PROFIBUS standard or in the ET 200 communication specifications. Their values have a significant effect on the timer run times requested with the "set_timer ()" function.
The entries are made in 100-µsec units for *min_slave_interval* and in 10-msec units for *dx_control_interval*.
Min_slave_interval describes the minimum time between two data telegrams to a slave. When AMPRO-DPM is used with the ASPC2 PROFIBUS ASIC, adherence to this time is monitored directly by the ASIC. All other telegrams except data telegrams (e.g., Global_Control commands, diagnostic requests and so on) are not affected by this interval. The slave with the longest *min_slave_interval* determines the wait time for the entire system.
Dx_control_interval should be selected in accordance with the PROFIBUS-DP standard to give the master sufficient time to complete at least six complete data cycles with all parameterized slaves. A new Global_Control telegram with the current master status (MA_OPERATE or MA_CLEAR) is sent when half the interval has passed. In addition, the "Data_Transfer_List in the master diagnosis is updated with this time interval. See above.

ts:

This parameter must contain the PROFIBUS station address of the master.

db_len_1:
db_len_2:
db_len_3:
db_len_4:

Since AMPRO-DPM uses the memory management of AMPRO2, the USER must have initialized this memory management system before AMPRO-DPM is started. See page 75 ff.
AMPRO2 memory management supplies several fixed-length block types for data (data blocks = DB) and free memory management for block sizes which vary with the program run time. Since the variable blocks cannot be enabled again, only fixed block sizes are used for AMPRO-DPM.
Up to four types which can be identified by number are available for the DB. Only their lengths vary. The USER specifies the lengths and uses this specification to initialize memory management. In addition, the USER informs AMPRO-DPM of his choice in DPMIB. AMPRO-DPM then uses the "l2_mem_alloc_dbx ()" AMPRO2 functions to allocate a DB of the smallest possible block type for the amount of data available per block. The *x* in the function name stands for the block type number. If a block size is not used, the USER must specify a length of 0_H for its type number.
Since the USER himself selects the lengths, he can also adapt the block size to the current parameterization (i.e., number of slaves, type and setup) and to available memory in the user system.

sync_mask:
freeze_mask:

These two entries specify which of the possible groups are SYNC groups and which are FREEZE groups for a Global_Control_Command. See the "set_slave_mode ()" function. Bit 0 corresponds to group 0 and so on. When the applicable bit is set for a group in both the SYNC mask and the FREEZE mask, this is a group for both commands. Groups with two cleared bits are not parameterized.

l2_apb_func_no:
l4_apb_type1_func_no:
l4_apb_type2_func_no:

In addition to the data blocks, AMPRO-DPM requires three different sizes of application blocks (APB; see section on memory resource requirements starting on page 75) from AMPRO2 memory management. The expected numbers of APBs for each size depend on the number, type and operating modes of the configured slaves. To save memory space, total memory space available for APBs can be optimized to a special application, or, even better, be designed to accommodate the maximum number of APBs which can occur.

AMPRO2 memory management can handle up to two different APB sizes. To permit the USER to optimize definition (i.e., use as little space as possible) of his APB sizes and their allocation to the "l2_mem_alloc_apb1 ()" or "l2_mem_alloc_apb2 ()" function, AMPRO-DPM expects the number of the APB allocation function in the variables *l2_apb_func_no*, *l4_apb_type1_func_no* and *l4_apb_type2_func_no* with which the applicable APB must be obtained by AMPRO2. A possible allocation is shown below.

APB sizes:
- APB, size 1: 26 bytes for L2-APBs and for type-1 L4-APBs
- APB, size 2: 28 bytes for type-2 L4-APBs

Entries in the variables:
- l2_apb_func_no: 1
- l4_apb_type1_func_no: 1
- l4_apb_type2_func_no: 2

l4_apb_type1_opcode_mask:
l4_apb_type2_opcode_mask:

AMPRO2 offers up to three different L4 header lengths for the use of L4 APBs (i.e., a fixed length of 2 bytes and two variable lengths). The latter must be declared when the FLC_FMA_MAC_RESET AMPRO2 function is called. AMPRO-DPM requires the 2-byte (L4 APB, type 1) and 4-byte (L4 APB, type 2) lengths.
During running operation, AMPRO2 uses various masks which are added to the normal opcode to determine when it is necessary to use data added to the APB as L4 headers. These masks are defined in the "\COMMON\L2_USER.H" file as shown below.

- Fixed L4 header size: L2_L4_HLEN_FIXED
- Variable L4 header size 1: L2_L4_HLEN_VAR1
- Variable L4 header size 2: L2_L4_HLEN_VAR2

To allow the USER as much freedom as possible when designing his system, AMPRO-DPM does not prescribe the masks to be used. They can be transferred when AMPRO-DPM starts up. The USER enters the AMPRO2 masks in the variables *l4_apb_type1_opcode_mask* and *l4_apb_type2_opcode_mask.* These masks have the 2 and 4-byte lengths as specified by the declarations in the FLC_FMA_MAC_RESET function. An example is shown below.

Declaration with call of the FLC_FMA_MAC_RESET AMPRO2 function:
- Fixed L4 header size 1: 2 bytes (cannot be used and thus cannot be parameterized)
- Variable L4 header size 1: 8 bytes (used by other components when necessary)
- Variable L4 header size 2: 4 bytes (also used by AMPRO-DPM)

Entries in DPMIB:
- l4_apb_type1_opcode_mask: L2_L4_HLEN_FIXED
- l4_apb_type2_opcode_mask: L2_L4_HLEN_VAR2

special_functions:

The individual bits of this byte can be used to trigger functions of the master which are implemented in AMPRO-DPM in addition to the services required by the DP standard. Currently, only bits 0 to 3 are required. Bits 4 to 7 are reserved for later developments. The layout of the byte is shown below.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0_B	0_B	0_B	0_B	LCCO	AÜDNA	NoTest	ASTP

ASTP:

This bit switches the AUTOSTOP master operating mode on (ASTP = 1_B) or off (ASTP = 0_B). When this mode is activated, slaves which exit the DATA state are removed from the processing cycle when a change in state from DATA or DIAG2_STATUS via PRM_UNLOCK to STOPPED occurs. See "state_report ()" CBF starting on page 129. The USER must then call the "restart_slave ()" function when the station is to be processed again. See page 103 ff.

NoTest:

Bit 1 can be used to select whether AMPRO-DPM is to use a dummy job to test whether the bus is available (bit 1 = 0) during startup. When bit 1 = 1, the test is omitted. In this case, the "init ()" function is always concluded synchronously with the return value DPM_OK, and the "init_done ()" CBF is not called. Caution: If the USER disables the test, he must ensure that a functional bus is available for AMPRO-DPM. Applicable conditions include: The master has already been accepted in the ring as an active station. The bus does not have a short circuit. And so on.

AÜDNA:

The DP standard states that a slave in the DATA state does not receive a response from the master as long as polling with data telegrams is being performed and the slave has not reported again with or without errors. The status of the slave entered by AMPRO-DPM in this case: DATA_NA. See the "state_report ()" CBF starting on page 129. After a message without errors, data communication is resumed immediately. After a message with errors, a branch is made to DIAG1 or DIAG2_STATUS depending on the type of error.
This also takes place when a bus short circuit occurs. When the RS 485 interfaces are floating or RS 485 driver blocks of different capacities are used (e.g., due to age), the master may not receive the responses of the slave although the slave receives all telegrams from the master. In the case described above, the outputs of the slave remain set although the master recognized a slave malfunction since trigger monitoring (ASÜ) of the slave does not expire. This can endanger the safety of the system.
To solve this problem, AÜDNA = 0_B can be selected as a form of slave handling which deviates from the DP standard. For slaves for which the ASÜ is enabled, all errors in the states PRM, CFG, DIAG2, DIAG2_STATUS and DATA cause an immediate transition to PRM_UNLOCK and then to DIAG1. These slaves never reach the DATA_NA state. For slaves without ASÜ, processing remains the same as that defined in the DP standard. Bit AÜDNA = 1_B must be set when the procedure causing safety risks is to be used in accordance with the DP standard for all slaves. We strongly advise against this.
This behavior does not occur when the AUTOSTOP function is used since the described principle is already used in the core. Setting of the AÜDNA bit is irrelevant in this case.

LCCO:

When slaves are operated with "long" consistency or in Buffered_Mode, the current output data buffer which is sent to the slave cannot be accessed by the USER via the SLCB. This makes it impossible for the USER to clear the current buffer (e.g., when a slave malfunction occurs or when master mode changes to CLEAR). When the slave returns to the data state again (DATA, DATA_NA), the data which were provided before are sent to the slave. From the viewpoint of the USER, these data may be out of data, however.
To prevent this, the USER can use LCCO = 1_B (**L**ong**C**onsistency**C**lear**O**utputs) to automatically clear the current output data buffer in the above described cases. When the slave returns to the data state, it first receives the data telegrams which were cleared. When the "output_update ()" function (see page 121 ff.) is called by the USER, the new data are sent in conclusion.

asic_type:

Some of the AMPRO2 features used by AMPRO-DPM can only be used with certain types of PROFIBUS ASICs. Currently, only the ASPC2 PROFIBUS ASIC can be used for AMPRO-DPM. This ASIC is available in two versions (i.e., STEP B and STEP C). AMPRO2 can determine which ASIC STEP is being used. The USER receives one of the two AMPRO2 definitions ASPC2_STEP_ID_B or ASPC2_STEP_ID_C as the return value. The USER must transfer this exact value to AMPRO-DPM in the *asic_type* variable. Depending on this setting, AMPRO-DPM uses or does not use the special functions of the later release of the ASPC2 (i.e., STEP C).

master_class:

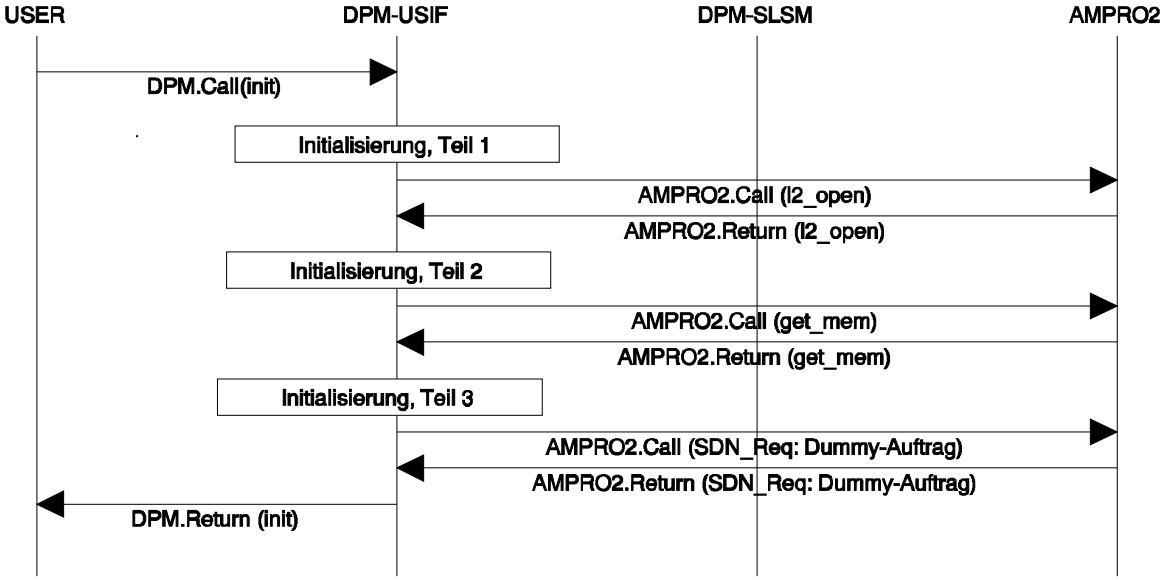
This parameter controls how AMPRO-DPM evaluates responses to DIAG1 telegrams. A class-1 master usually terminates an attempt to contact a slave when the slave reports in the diagnosis that it has already been locked by another master. The slave continues to receive DIAG1 telegrams until the other master releases the slave again. Only after this release may the class-1 master access the slave. The *master_class* parameter must be initialized with the value DPM_MASTER_CLASS_1 to achieve this normal behavior. See the COMMON\DPM_COMM.H file.
When AMPRO-DPM is used as a class-2 master, this master must usually check correct parameterization of all slaves. The class-2 master attempts to contact all slaves briefly. The slaves are enabled again after the check. To ensure that this procedure can be performed quickly, AMPRO-DPM does not wait for another

master to release the slave and ignores the Master_Lock entry of the diagnosis, thus "capturing" the slave for the class-2 master. The *master_class* parameter must be initialized with the value DPM_MASTER_CLASS_2 to obtain this behavior.

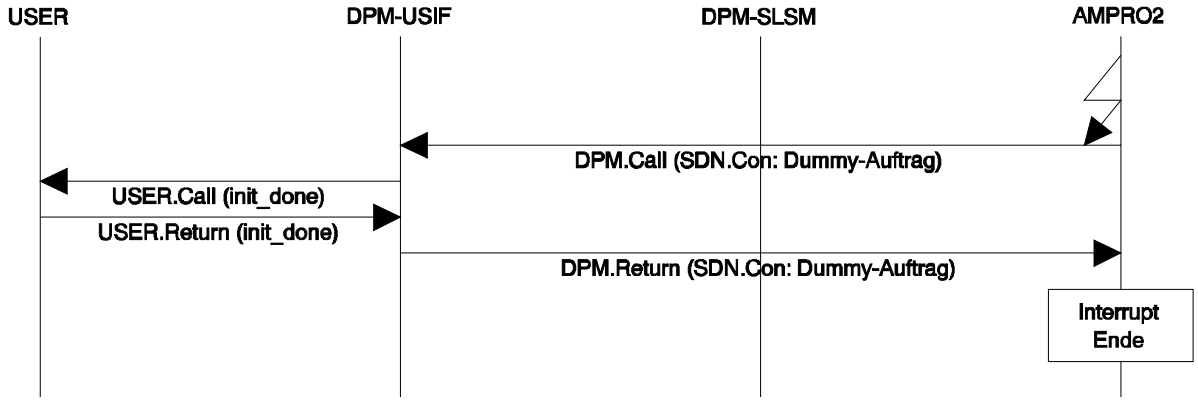
5.2.3.2.3 Call

Sequence chart:

Calling the function, initialization and starting the dummy job



Receipt of confirmation of the dummy job and response message to the USER



Sample call:

```
status = dpm_ptr -> init ((DPMIB_PTR) &dpmib);
```

Possible return values for *status*:

DPM_OK_CBF	The job was started correctly. Call of the CBF indicates the end of execution.
ERR_DPM_INIT_DONE	Error: The "init ()" function has already been started since the last "dpm_open ()" call.
ERR_DPM_REQ_ACTIVE	Error: The last "init ()" call is still being processed.

ERR_DPM_MEM_ALLOC	Error:	Not enough memory space is available.
ERR_DPM_WRONG_ASIC_TYPE	Error:	Neither the define value ASPC2_STEP_ID_B nor the value ASPC2_STEP_ID_C was entered as the ASIC type.
ERR_DPM_INVALID_PAR	Error:	Wrong parameter in the USER request

5.2.3.3 add_slave (SLCB_PTR)

5.2.3.3.1 Description

The "add_slave ()" function causes AMPRO-DPM to set up the internal management structures for the slave specified by the transferred SLCB. This can be done in all master states (MA_STOP, MA_CLEAR, or MA_OPERATE). In the master state MA_CLEAR or MA_OPERATE, AMPRO-DPM also attempts to include the slave in the processing cycle, and, if possible, make contact with the slave. This function is called once per slave.
The slave control block of the slave (must be completed correctly beforehand) is used as the transfer parameter. The initial operating mode of a slave is always DEACT. This applies until the first call of the "state_report ()" function reports a new state. This new state is usually the transition from DEACT ➔ DIAG1. See "state_report ()" function starting on page 128. During each call, AMPRO-DPM fetches the required number of application and data blocks from AMPRO2 memory management and completes them accordingly.
After "add_slave ()", communication continues with interrupt control. During normal operation, once a slave has been activated, it must be removed from the processing cycle with "withdraw_slave ()" (see page 100 ff.) when further processing is to be suppressed regardless of the state of the master. When the AUTOSTOP master operating mode is used (see *special_functions* entry in DPMIB starting on page 88), the slave is automatically deactivated by AMPRO-DPM when necessary. The slave is restarted with the "restart_slave ()" function. See page 103 ff. All management information for the slave is retained in this state. When the slave is to be permanently removed from the processing cycle (i.e., management information is also cleared), the "withdraw_slave ()" function must be called even in AUTOSTOP master operating mode.

5.2.3.3.2 Call

Sample call: Slave no. x is to be included.

```
status = dpm_ptr -> add_slave ((SLCB_PTR) &slcb[x]);
```

Possible return values for *status*:

DPM_OK		Service transferred correctly to AMPRO2
ERR_DPM_REQ_ACTIVE	Error:	The last "init ()" call is still being processed.
ERR_DPM_MEM_ALLOC	Error:	Not enough memory space is available.
ERR_DPM_INVALID_PAR	Error:	Wrong parameter in the USER request
ERR_DPM_UNKNOWN_TYPE	Error:	When a previously unknown slave type identifier from 0 to 127 is specified, this slave is treated as a normal standard DP slave. Unknown identifiers from 128 to 255 are acknowledged with this error return value.

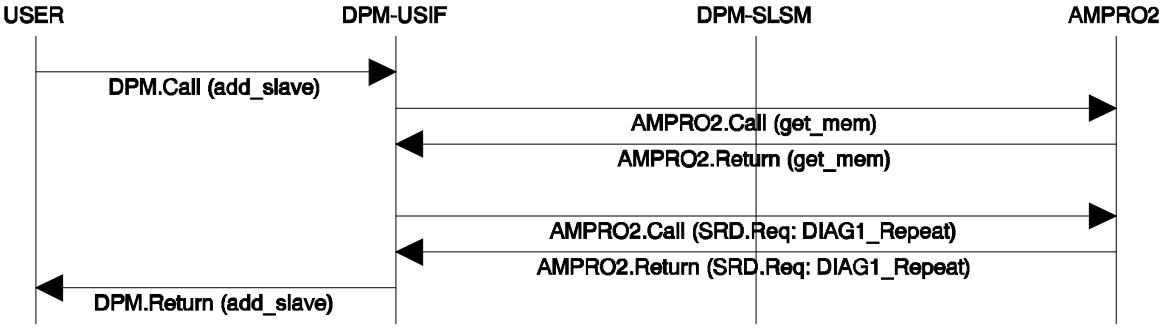
5.2.3.3.2.1 Processing for the Slaves Parameterized for the Master

The following sequence charts show the start of the processing cycle for a standard DP slave which reacts immediately and correctly. In addition, the slave must be parameterized as a slave of this master. Previous ET 200 slaves (i.e., DP Siemens slaves) are covered here too but the telegram sequence and the layout of

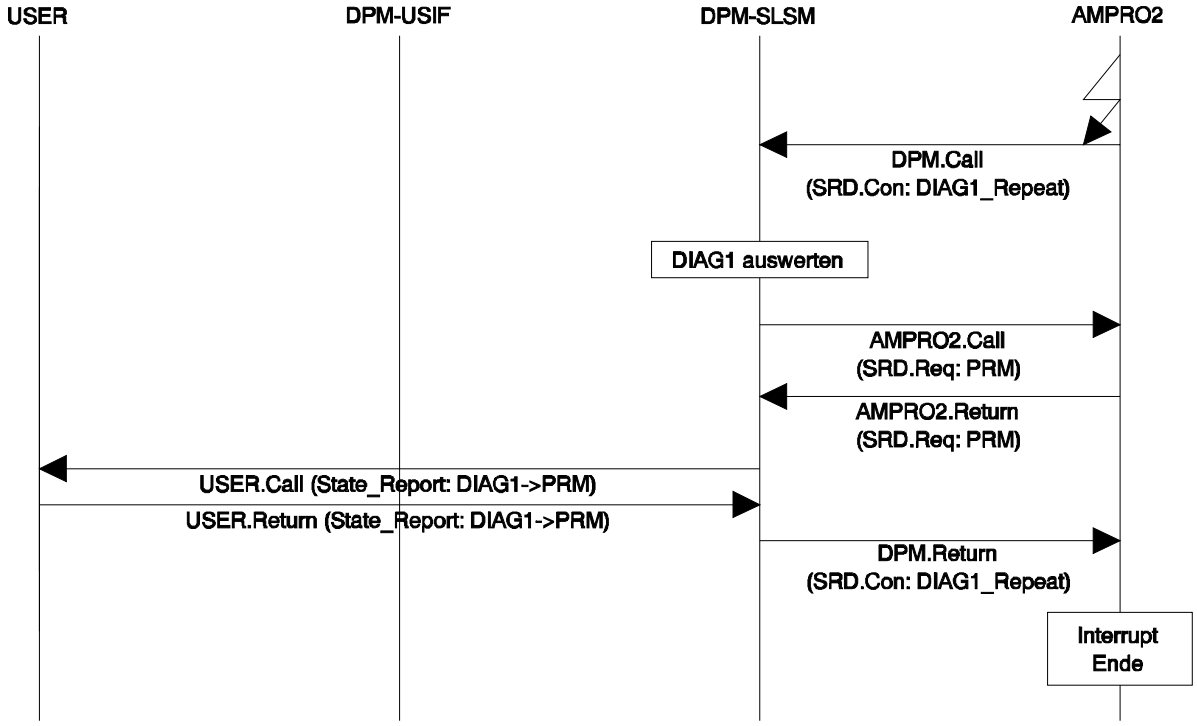
the individual telegrams changes depending on the type of slave. For an overview of all possible status transitions for all types of slaves, see the section on slave families supported by AMPRO-DPM starting on page 145.

Sequence charts:

Call of the function by the USER and request first diagnosis (DIAG1)

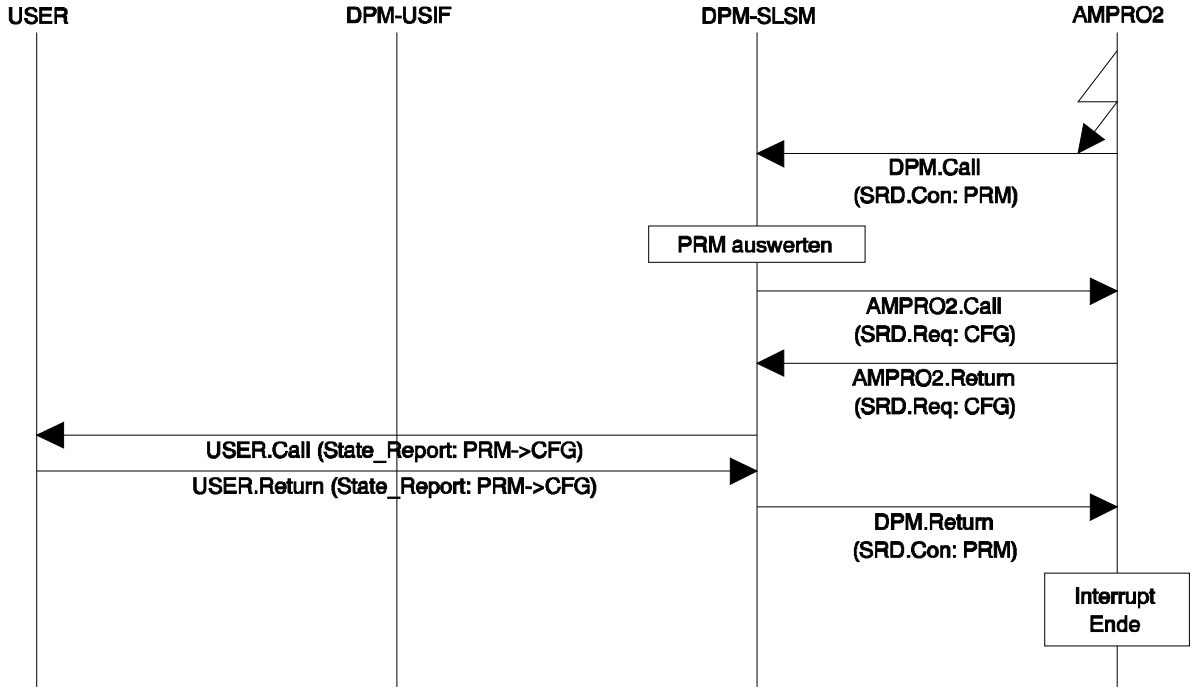


Evaluate confirmation to DIAG1 and send parameterization (PRM)

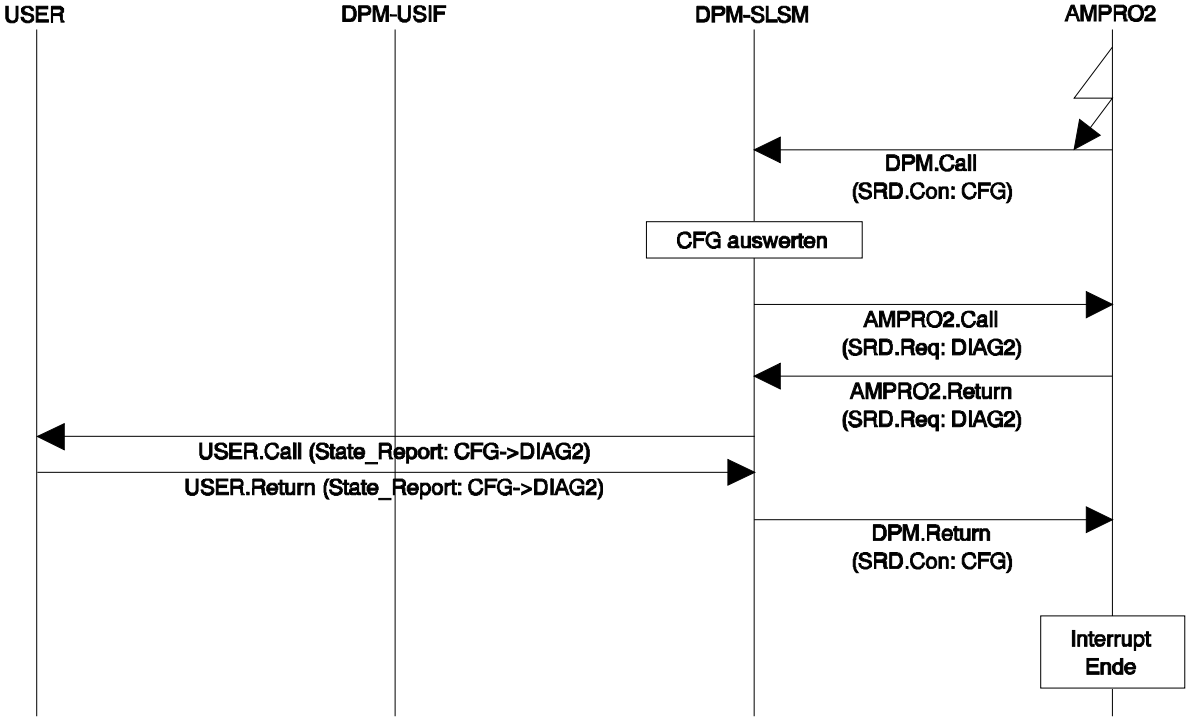


 01/09

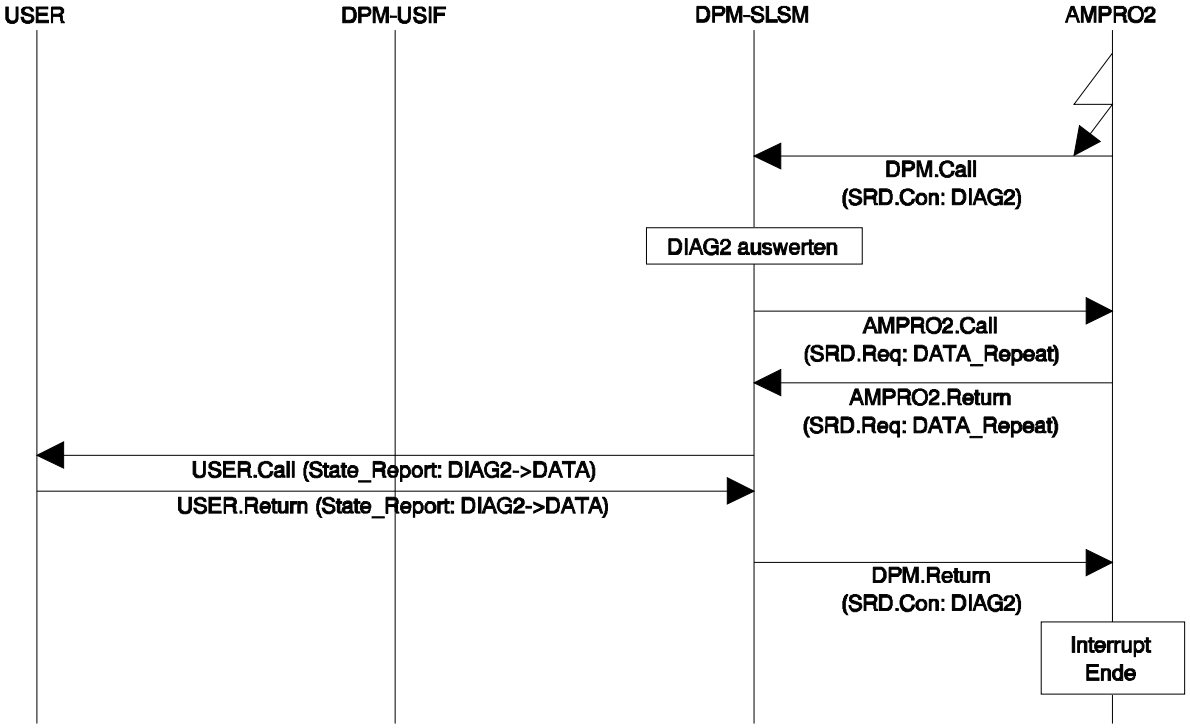
Evaluate confirmation to PRM and send parameterization (CFG)



Evaluate confirmation to CFG and request second diagnosis (DIAG2)



Evaluate confirmation to DIAG2 and perform data communication (DATA) with the slave



The standard DP slave is now in the data cycle (i.e., continuous process data communication is taking place). The master does not process the slave again unless consistent data are involved or an error occurs. This procedure adheres to the procedure for error treatment described in the PROFIBUS-DP standard or the procedure for consistent data described in these specifications.

5.2.3.3.2.2 Processing of Slaves Not Parameterized for the Master (Shared_Input Slaves)

In addition to the form of communication described above, standard DP slaves provide the capability of allowing several masters to read the inputs of the slave. Prerequisite is that one of the masters has processed the slave as described above. This master is the main or parameterization master of the slave.
When the slave assumes the DATA state with the parameterization master, all other masters can read the inputs of this slave as auxiliary or Shared_Input masters. However, only the parameterization master can read and set the outputs. An entry in the parameter record of the slave determines which master is the parameterization master. See parameter module description starting on page 158. When the current master is only an auxiliary master, it processes the slave with special data telegrams. See Read_Input on page 135. Separate designators are defined for the states in which the slave sends special Shared_Input telegrams. See "state_report ()" function starting on page 128.

5.2.3.4 withdraw_slave (SLCB_PTR)

5.2.3.4.1 Description

The "withdraw_slave ()" function can be used by the USER to deactivate a slave which was previously transferred to AMPRO-DPM with "add_slave ()". This function can be called in all master states (i.e., MA_STOP, MA_CLEAR or MA_OPERATE).
After a "withdraw_slave ()" call, AMPRO-DPM must stop the slave handler of the slave specified with the transferred SLCB. See "state_report ()" CBF starting on page 129. If the slave is not in the DIAG1, STOPPED or PRM_UNLOCK state, AMPRO-DPM initiates the transition of the slave state to PRM_LOCK to log itself off on the slave. The slave assumes the DEACT state. The transition to DEACT occurs immediately from the states DIAG1, STOPPED or PRM_UNLOCK. The USER is informed of each of these transitions in state by "state_report ()". The slave handler is stopped. AMPRO-DPM releases all APBs and DBs which were allocated by AMPRO2 memory management for the slave with "add_slave ()", and the SLCB is removed from internal AMPRO-DPM management.
Depending on the current state of the master and the slave, this function can be processed either synchronously or asynchronously. The USER recognizes this by the different return values (i.e., DPM_OK for synchronous processing and DPM_OK_CBF for asynchronous processing). The "withdraw_slave_done ()" CBF (see page 136 ff.) is only called after the DPM_IOK_CBF return value as the return message on the conclusion of the "withdraw_slave ()" function. Otherwise, the function has already been concluded at the end of the function call.

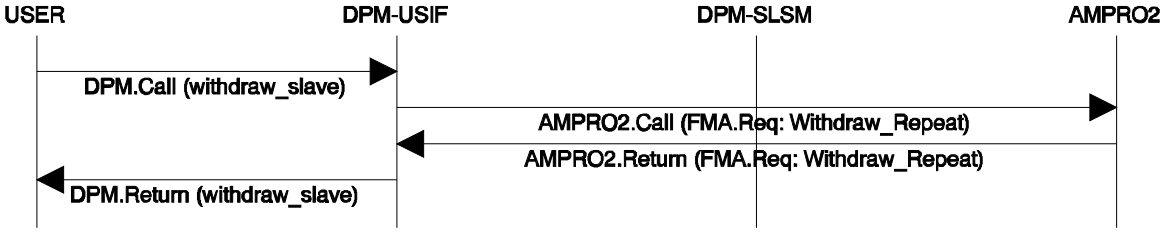
When the function has been processed and, if necessary, the CBF has been concluded, sole access rights to the SLCB (as before "add_slave ()" was called) are returned to the USER. In addition to simple deactivation, the USER can now change the parameter record of the slave, or update it and transfer it again to AMPRO-DPM with "add_slave ()". See page 96 ff. In addition, "withdraw_slave ()" must be called for every activated slave when AMPRO-DPM is to be stopped completely with the "close ()" function. See page 126 ff.

5.2.3.4.2 Call

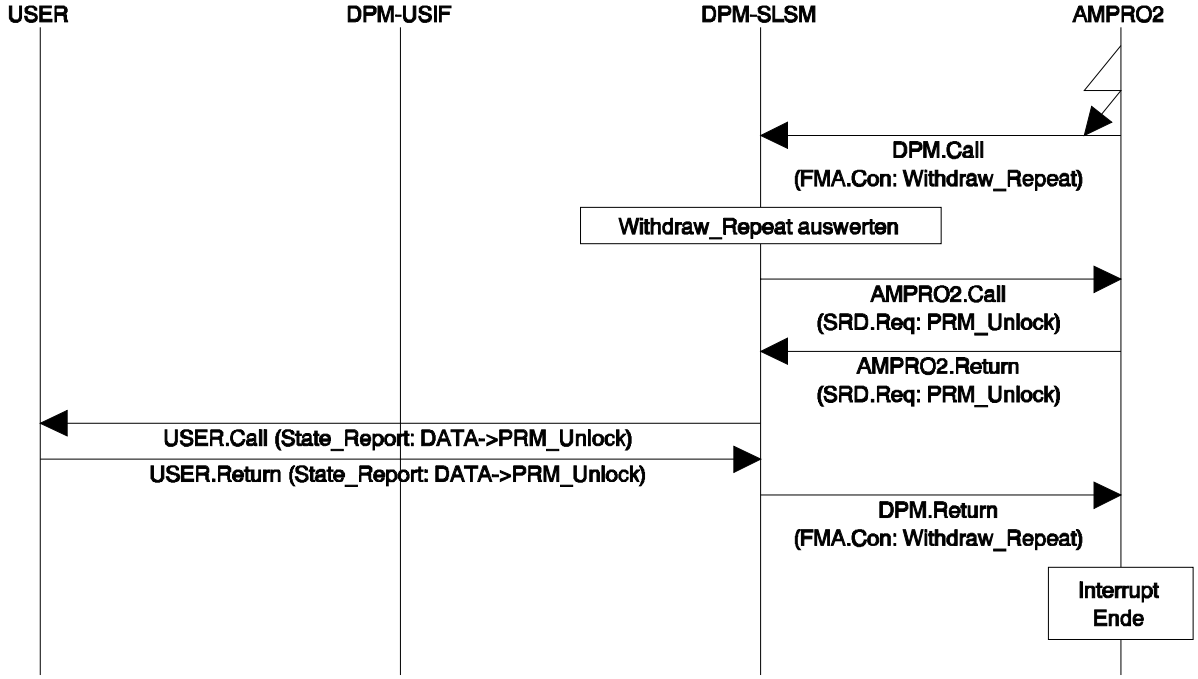
Sequence charts:

In these sequence charts, slave no. x is already in the data cycle. The procedure for other states conforms to the PROFIBUS-DP standard or the ET 200 communication specifications (e.g., wait for return of the last diagnostic job instead of issuing the Withdraw_Repeat job). For state transitions, see also "state_report ()" starting on page 129.

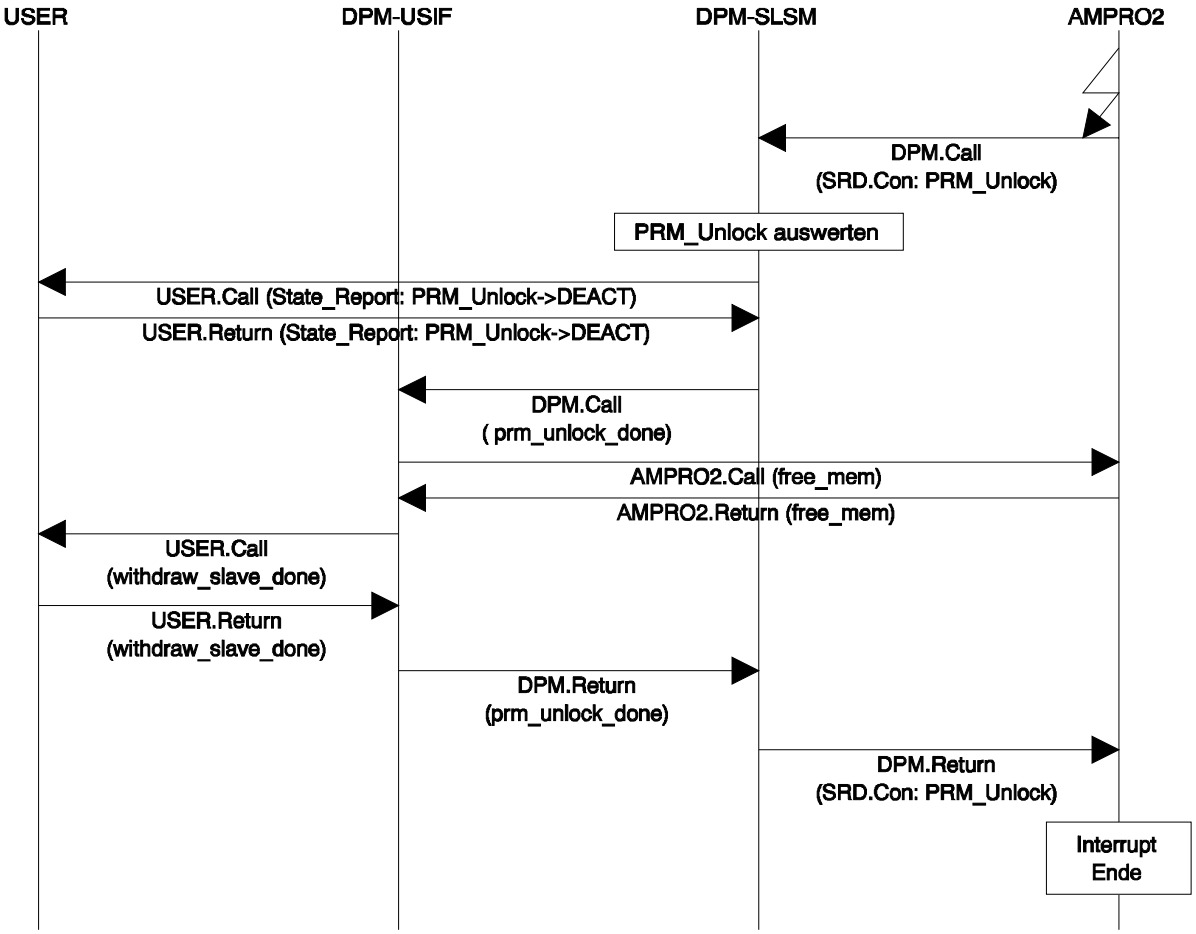
Function called by the USER. Start AMPRO-DPM job "Withdraw_Repeat":



Evaluate confirmation for "Withdraw_Repeat" and send "PRM_Unlock" to the slave



Evaluate confirmation for "PRM_Unlock" and inform USER



Sample call: Slave no. x is to be removed from the processing cycle.

```
status = dpm_ptr -> withdraw_slave ((SLCB_PTR) &slcb[x]);
```

Possible return values for *status*:

DPM_OK	The complete job has been executed correctly. The CBF is no longer called.
DPM_OK_CBF	The job was started correctly. The end of execution is indicated by the CBF call.
ERR_DPM_REQ_ACTIVE	Error: The last call of the "init ()" function or the "withdraw_slave ()" function is still being processed.

5.2.3.5 restart_slave (SLCB_PTR)

5.2.3.5.1 Description

The "restart_slave ()" function can only be called when AUTOSTOP master mode has been activated. See section on DPMIB starting on page 88. In this mode, slaves which exit the DATA or DIAG2_STATUS state are removed from the processing cycle with a change in state via PRM_UNLOCK to STOPPED (see "state_report" CBF starting on page 129). The STOPPED slave state can also only be reached in AUTOSTOP master mode. This master mode is recommended when the USER reaction to a station failure requires substantially more time than the processing cycle on the bus. When the slave assumes STOPPED status, the USER can take his time when reacting to this event. After conclusion of USER processing, the USER transfers the SLCB of the slave via "restart_slave ()" back to the AMPRO-DPM which will process the slave when the DIAG1 state is assumed (i.e., similar to "add_slave ()").
The USER could obtain similar behavior by starting the "withdraw_slave ()" function after the "state_report ()" CBF has been called for a slave which has just exited the DATA or DIAG2_STATUS state. After processing and evaluating the diagnoses, the USER could start up the slave handler again with "add_slave ()". The disadvantage of this procedure is that each "withdraw_slave ()" would cause the entire internal AMPRO-DPM management of the slave (i.e., conditional allocation of APBs and DBs, initialization of the blocks and the internal lists and variables) to be cleared and each "add_slave ()" would cause it to be set up again. If, as in this case, the USER only wants to deactivate the slave handler temporarily and not stop the slave completely, the continuous set up and clearing of the management information serves no purpose and is also very time consuming. The AUTOSTOP master mode offers decisive advantages here.
"restart_slave ()" can only be called in AUTOSTOP master mode and only when a slave which was transferred to AMPRO-DPM before with "add_slave ()" is in STOPPED status. As with all other transitions, USER accesses to the SLCB are only permitted while the "state_report ()" CBF is being processed. If necessary, the USER must use internal flags for the rest of his program. See also section on communication model starting on page 61. During the time between which the "state_report ()" CBF is called with slave transition to STOPPED and the "restart_slave ()" function is called by the USER, the slave is not addressed via the bus but AMPRO-DPM must occasionally access the SLCB to update certain entries. For this reason, the SCLB can be maintained in this slave status in an internal AMPRO-DPM list. A slave in STOPPED status can also be deactivated with "withdraw_slave ()" without the USER having to call "restart_slave ()" beforehand.

5.2.3.5.2 Call

Sequence charts:

Processing of the "restart_slave ()" function call is the same as processing of the "add_slave ()" call. See the sequence charts of "add_slave ()".

Sample call: Slave no. x is to be started again.

```
status = dpm_ptr -> restart_slave ((SLCB_PTR) &slcb[x]);
```

Possible return values for *status*:

DPM_OK	Service transferred correctly to AMPRO2	
ERR_DPM_WRONG_SLAVE_STATE	Error:	The slave is not in SL_STOPPED status.
ERR_DPM_REQ_ACTIVE	Error:	The last call of the "init ()" function or the "restart_slave ()" function is still being processed.
ERR_DPM_SLAVE_IS_PROCESSED	Error:	To prevent user errors, the "restart_slave ()" function can reject a second call of the function for the same slave more frequently than before.

5.2.3.6 suspend_slave (SLCB_PTR)

5.2.3.6.1 Description

Calling the "suspend_slave" function is only recommended when AUTOSTOP master mode has been activated. See section on DPMC_IB. In this mode, slaves which exit the DATA or DIAG2_STATUS state are removed from the processing cycle with a change in state from there over PRM_UNLOCK to STOPPED. See "state_report ()" CBF starting on page 129. This master mode is recommended when the USER reaction to a station failure requires substantially more time than the processing cycle on the bus. When the slave assumes STOPPED status, the USER can take his time when reacting to this event.
In addition to the automatic transition to STOPPED, the USER can force this transition by calling the "suspend_slave" function. This is recommended when processing of one or more slaves is to be frozen after the USER detects an error in one of his components, for example. As with the automatic transition to STOPPED, a restart of the slave must be performed by calling the "suspend_slave" function. See page 103 ff.
The "suspend_slave" function cannot be called unless a slave which was previously transferred with "add_slave" is not in STOPPED status. Separate indication of the end of "suspend_slave" function execution is not required. As with automatic transition, the call of the "state_report ()" CBF with slave transition to STOPPED is usually used as the acknowledgment. As with all other transitions, the USER may only access the SLCB while the "state_report ()" CBF is being processed. If necessary, the USER must use internal flags for further processing of his program. See also section on communication model and the description of "restart_slave".
When the master is in STOP status, the slaves are usually in DEACT status. In this case, the USER can use the "suspend_slave" function to suspend one or more slaves in advance before the next change in state of the master to CLEAR or OPERATE. As an exception, the transition of the slave from DEACT to STOPPED takes place without the "state_report ()" CBF being called. The return value of the "suspend_slave" function tells the USER that this has happened.

5.2.3.6.2 Call

Sequence charts:

Processing of the "suspend_slave" function is very similar to processing of the "withdraw_slave" function. See sequence charts of the "withdraw_slave" function.

Sample call:

Sample call: Slave no. x is to be started again.

```
status = dpm_ptr -> suspend_slave ((SLCB_PTR) &slcb[x]);
```

Possible return values for *status*:

DPM_OK	Service executed correctly. The slave is in STOPPED status. No more calls of the "state_report ()" CBF will be made.
ERR_DPM_OK_CBF	Service transferred correctly to AMPRO2. Further processing can be taken from the transitions reported with the "state_report ()" CBF.
ERR_DPM_WRONG_SLAVE_STATE	Error: The slave is already in the SL_STOPPED state.
ERR_DPM_REQ_ACTIVE	Error: A request is already active.

5.2.3.7 set_master_mode (SMMCB_PTR)

5.2.3.7.1 Description

The "set_master_mode ()" function sets the operating mode of the master as specified by the transfer parameter, and treats all activated slaves as described in the PROFIBUS-DP standard. Possible states are MA_STOP, MA_CLEAR and MA_OPERATE.
The MA_OFFLINE state mainly pertains to the start of the layer-2 interface. Since the DP master would only be able optimize the start of AMPRO2 with its own interests in mind, this start must be performed by the user. For this reason, the MA_OFFLINE state is also not defined for AMPRO-DPM.
The default master mode is MA_STOP before "set_master_mode ()" is called for the first time. Depending on the type of change in state, only the return value of the AMPRO-DPM job is generated as the return message, or the "set_master_mode_done ()" USER CBF may also be called.

The first call of the "set_timer ()" CBF (see page 140 ff.) always takes place during the first USER call of the "set_master_mode ()" function. The CBF call is usually omitted for all subsequent calls of the USER function.

The "add_slave ()" and "withdraw_slave ()" functions can be called in any master state. However, cyclic data communication does not start until the MA_OPERATE state is assumed. A direct change from MA_STOP to MA_OPERATE or from MA_OPERATE to MA_STOP is not possible. This transition must always take place over the MA_CLEAR state. Even when the master changes states, all slaves transferred with "add_slave ()" (see page 96 ff.) remain with AMPRO-DPM until they are withdrawn by the USER again with "withdraw_slave ()". See page 100 ff. The call of the "set_master_mode_done ()" CBF is delayed by at least one and not more than two data cycles for transitions from MA_CLEAR to MA_OPERATE or from MA_OPERATE to MA_CLEAR. This ensures that all slaves are informed of the last change in state of the master.

DP Siemens slaves cannot be set to CLEAR mode with the Global_Control command MA_CLEAR. A Singlecast telegram (SDN high without data) is required for each of the DP Siemens slaves with the SPM or SPC ASIC as called for in the ET 200 communication specifications. This telegram is sent to each DP Siemens slave. The USER is not aware of this special treatment, however.

The DP standard defines CLEAR behavior of the class-1 master as follows. While this master is in MA_CLEAR status, it must send all slaves normal data telegrams with user data zeroed out. In addition to this procedure and beyond the specifications of the DP standard, the slave can be sent a data telegram with no user data at all while the master is in MA_CLEAR status, instead of a normal data telegram. The master does not begin sending data telegrams with current data again until a change in state to MA_OPERATE occurs. This FAILSAFE (FASA) procedure can only be used for slaves which can also be parameterized as FAILSAFE (FASA) slaves. By making appropriate entries in the slave parameter record, the USER causes AMPRO-DPM to use this procedure for certain slaves. See page 174.

The following table shows a list of possible AMPRO-DPM reactions.

Possible changes in state:

New State \ Previous State		STOP	CLEAR	OPERATE
STOP	Action	None	Deactivate all existing slave stations	None
	Return value of the DPM function	DPM_OK	DPM_OK_CBF	ERR_DPM_NOT_ ALLOWED
	Call of the call back function	No	After execution	No
CLEAR	Action	Activate all existing slave stations	None	Issue first "Clear" global control command
	Return value of the DPM function	DPM_OK	DPM_OK	DPM_OK_CBF
	Call of the call back function	No	No	After execution
OPERATE	Action	None	Issue first "Operate" global control command	None
	Return value of the DPM function	ERR_DPM_NOT_ ALLOWED	DPM_OK_CBF	DPM_OK
	Call of the call back function	No	After execution	No

5.2.3.7.1.1 set_master_mode Control Block (SMMCB) Header

Type	Designation	Direction of Access			
		USER		DPM	
		L	S	L	S
DPM_IFA_DATA_ATTR *	next_blk_ptr			X	X
DPM_IFA_DATA_ATTR *	prev_blk_ptr			X	X
Unsigned8	opcode (DPM_SET_MASTER_MODE)	X	X	X	
Unsigned8	subsystem (dpm_handle)	X	X	X	
L2_TYPE_ID_PTR	id_ptr (Can be used by the USER as desired)	X	X		

The header is filled out the same as the header description of the SLCB (see page 82 ff.) except that the function identifier DPM_SET_MASTER_MODE (see page 156) must be entered for *opcode* here and the handle supplied by the "dpm_open ()" function (see page 86 ff.) must be entered for *subsystem*.

5.2.3.7.1.2 General Portion

Type	Designation	Direction of Access			
		USER		DPM	
		L	S	L	S
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to "set_master_mode_done ()" CBF	X	X	X	
Unsigned8	Status (MA_STOP; MA_CLEAR; MA_OPERATE)	X	X	X	
Unsigned8	*Reserved*				

Pointer to "set_master_mode_done ()" CBF :

This pointer must point to the "set_master_mode_done ()" CBF of the USER. See page 136.

status:

The new master status desired (A_STOP, MA_CLEAR or MA_OPERATE) must be entered by the USER here. See page156.

5.2.3.7.2 Call

Sample call:

```
status = dpm_ptr -> set_master_mode ((SMMCB_PTR) &smmcb);
```

Possible return values for *status*:

DPM_OK	The complete job was executed correctly and the CBF is no longer called.
DPM_OK_CBF	The job was started correctly. CBF call indicates end of execution.
ERR_DPM_REQ_ACTIVE	Error: The last call of the "init ()" function or the "set_master_mode ()" function is still being processed.
ERR_DPM_NOT_ALLOWED	Error: The requested change in state cannot be performed from the current state of the master.

5.2.3.8 set_slave_mode (SSMCB_PTR)

5.2.3.8.1

For more precise specification of this service, the user must transfer a pointer to the correctly completed set_master_mode control block (SMMCB structure; definition in the "\COMMON\DPM_COMM.H" file). This block is required once.

5.2.3.8.2 Description

The "set_slave_mode ()" command sets the slave mode (i.e., SYNC or UNYSNC and FREEZE or UNFREEZE) for one or more of the parameterized groups of slaves. AMPRO-DPM performs this function once. The group or groups of slaves remain in this mode until the mode is specified again with another

"set_slave_mode ()" command. When execution of the service is complete, the USER receives the SSMCB together with a status message from the "set_slave_mode_done ()" CBF. See page 138.
Although the interface for the USER is the same for all operating mode parameters, AMPRO-DPM handles the SYNC/UNSYNC parameter differently than the FREEZE/UNFREEZE parameter. The sequence charts in the following sections show these differences.
It is **not** necessary for the USER to make sure (e.g., with the "mark_cycle ()" function) that there is at least one data cycle before or after "set_slave_mode ()" is called. AMPRO-DPM ensures that at least one data cycle was processed for each call before the USER receives the job acknowledgment via "set_slave_mode_done ()".

This command can only be executed for standard DP slaves. In addition, all desired slaves must be assigned to one of up to eight groups. Each group is used to specify only the SYNC/UNSYNC command, only the FREEZE/UNFREEZE command or both commands. All of the slaves assigned to a group must also be able to process each of the group services.
In contrast to other commands, "set_slave_mode ()" can be processed up to eight times simultaneously. This requires a new SSMCB for each call to be processed simultaneously. The SSMCBs are transferred to AMPRO-DPM with one of the later calls of "set_slave_mode ()". To determine which block the USER has received back with "set_slave_mode_done ()", the combination group number/operating mode or the *id_ptr* entry in the header of the SSMCB can be used. In addition, the USER can also provide additional CBFs so that another CBF is called depending on the SSMCB.
In all jobs which are processed simultaneously, different bits should be set in each job for the group number since a standard DP slave may not receive two or even more "set_slave_mode ()" telegrams in such short succession without receiving data in-between. Correct processing of the "set_slave_mode ()" calls would still be endangered even if a slave sent a non permanent diagnosis before the "set_slave_mode ()" telegram. For state transition DATA → DIAG2_STATUS → DATA; see "state_report ()" CBF starting on page 129. To prevent the slave from outputting old data at the time of synchronization as could happen in the case above, the call of the "set_slave_mode_done ()" CBF must be delayed by at least two data cycles when at least one of the grouped slaves of a job is in the state DIAG2_STATUS.

5.2.3.8.3 set_slave_mode Control Block (SSMCB)

To specify this service in more detail, the user must transfer a pointer to the correctly completed set_slave_mode control block (definition: SSMCB) each time "set_slave_mode ()" is called. This block is required for all slaves as many times as simultaneous calls are to be processed. The number can be between one and eight blocks depending on the maximum number of calls.

5.2.3.8.3.1 Header

Type	Designation	Direction of Access			
		USER		DPM	
		L	S	L	S
DPM_IFA_DATA_ATTR *	next_blk_ptr			X	X
DPM_IFA_DATA_ATTR *	prev_blk_ptr			X	X
Unsigned8	opcode (DPM_SET_SLAVE_MODE)	X	X	X	
Unsigned8	subsystem (dpm_handle)	X	X	X	
L2_TYPE_ID_PTR	id_ptr (Can be used by the USER as desired)	X	X		

The header is completed the same as the header description of the SLCB (see page 82 ff.) except that the function identifier DPM_SET_SLAVE_MODE (see page 156) must be entered for *opcode* and the handle supplied by the "dpm_open ()" function (see page 86 ff.) must be entered for *subsystem*.

5.2.3.8.3.2 General Portion

Type	Designation	Direction of Access			
		USER		DPM	
		L	S	L	S
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to CBF "set_slave_mode_done ()"	X	X	X	
Unsigned8	remote_add (0_D-125_D; 127_D)	X	X	X	
Unsigned8	gc_command (SL_SYNC or SL_UNSYNC and SL_FREEZE or SL_UNFREEZE)	X	X	X	
Unsigned8	gc_group_select (01_H-FF_H; bit-coded)	X	X	X	
Unsigned8	*Reserved* (job_no)			X	X

Pointer to "set_slave_mode_done ()" CBF:

This pointer must point to the "set_slave_mode_done ()" CBF of the USER. See page 138.

remote_add:

When the current job is to apply to only one slave, the PROFIBUS address of this slave (0_D to 125_D) must be entered here. When a group of slaves is addressed, 127_D (i.e., global access address in accordance with standards) must be entered here. In both cases, a correct parameter is also required for *gc_group_select*. Even when only one slave is to be addressed, this slave will ignore the telegram when it is not included in the selected *gc_group_select* value.
There is no point in sending a "set_slave_mode ()" job to only one slave however, since this service is usually used to synchronize the data update cycle of several slaves.

gc_command:

The USER enters the function requested by him (i.e., SL_SYNC or SL_UNSYNC, or SL_FREEZE or SL_UNFREEZE) here. See section on rules of coding for slave operating modes on page 155.

gc_group_select:

Every standard DP slave for which the "set_slave_mode ()" command is to be executed must be assigned to a group by the USER. These groups contain slaves which can handle only SYNC/UNSYNC or only FREEZE/UNFREEZE or both. A slave which can handle both services can be assigned to a SYNC group but not vice versa. The two masks which the USER transferred with the "init ()" call in DPMIB (see page 88 ff.) determine which group handles which service. These masks are used to check the currently transferred value for plausibility.
The parameter is bit-coded (i.e., bit 0_B stands for group 0_D, bit 1_B stands for group 1_D and so on). 00_H (i.e., all slaves) is not permitted. FF_H (i.e., all slave groups) must be used instead.

Reserved (job_no):

This entry is required internally by AMPRO-DPM for management of the jobs. It is not applicable to the USER and may not be changed by him.

5.2.3.8.4 Call

Sample call:

```
status = dpm_ptr -> set_slave_mode ((SSMCB_PTR) &ssmcb);
```

Possible return values for *status*:

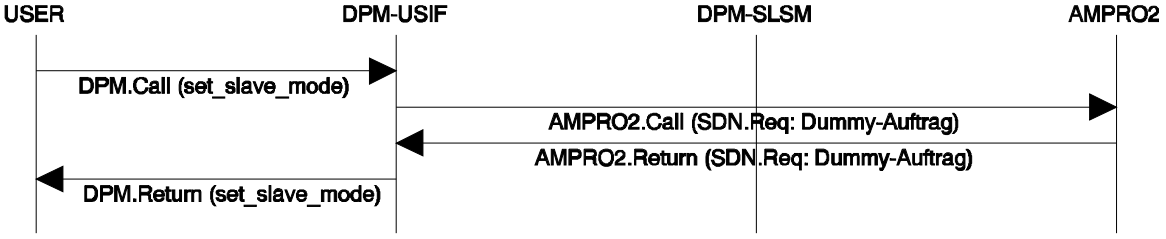
DPM_OK_CBF	The job was started correctly. The CBF call indicates end of execution.
ERR_DPM_REQ_ACTIVE	Error: The last call of the "init ()" function has still not been concluded, *or* the maximum number of simultaneous "set_slave_mode ()" calls was already reached with the last call. This job and all later jobs must wait at least until a running "set_slave_mode ()" job has been concluded.
ERR_DPM_INVALID_MODE	Error: Wrong master mode. Only possible in master mode MA_CLEAR or MA_OPERATE.
ERR_DPM_INVALID_PAR	Error: Wrong parameter in the USER request

5.2.3.8.4.1 SYNC/UNSYNC Slave Operating Modes

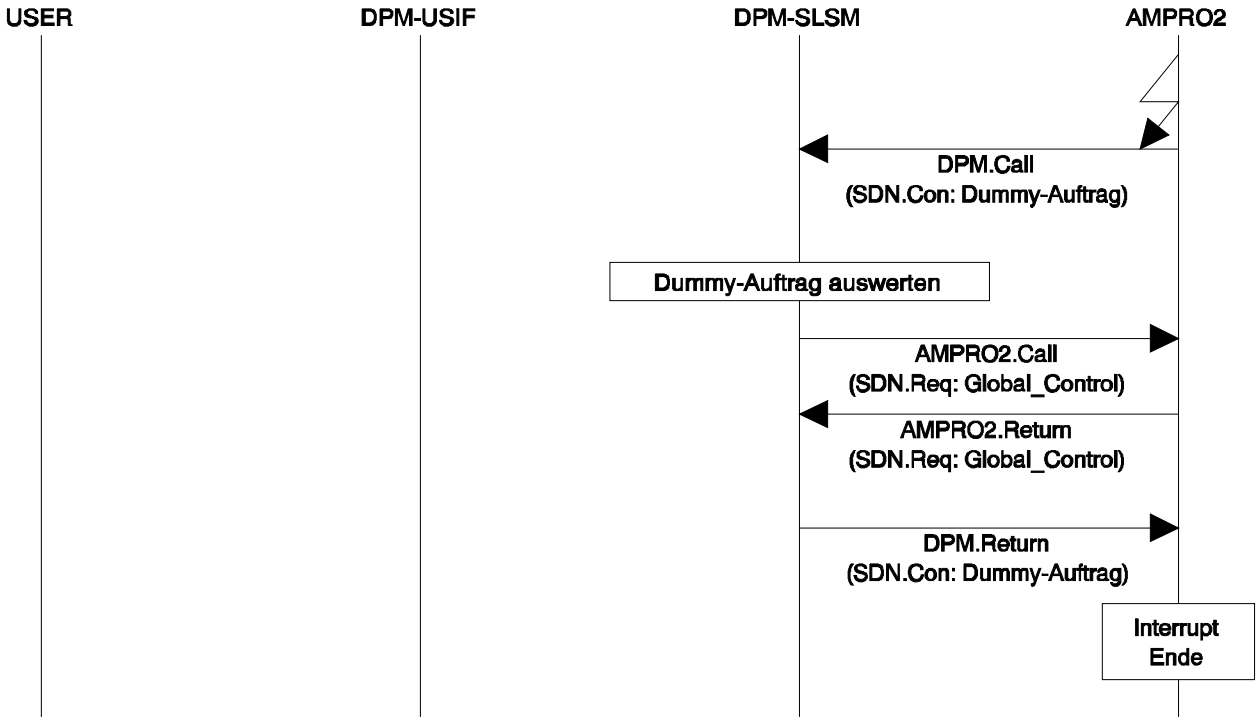
When the USER issues the SYNC command, the values entered last in the output data area must be frozen by the appropriate slaves. However, since "set_slave_mode ()" can be called immediately after the last change of the outputs, AMPRO-DPM ensures that all slaves receive at least one more data telegram before the SYNC Global_Control command is issued. The total delay time up to the issuing of the "set_slave_mode_done ()" return message is approximately the time required for one data cycle plus an additional single job.
During the time between which the "set_slave_mode ()" function is called and the response with the "set_slave_mode_done ()" function, the USER may no longer permit accesses to the output data areas of the addressed slaves.

Sequence charts:

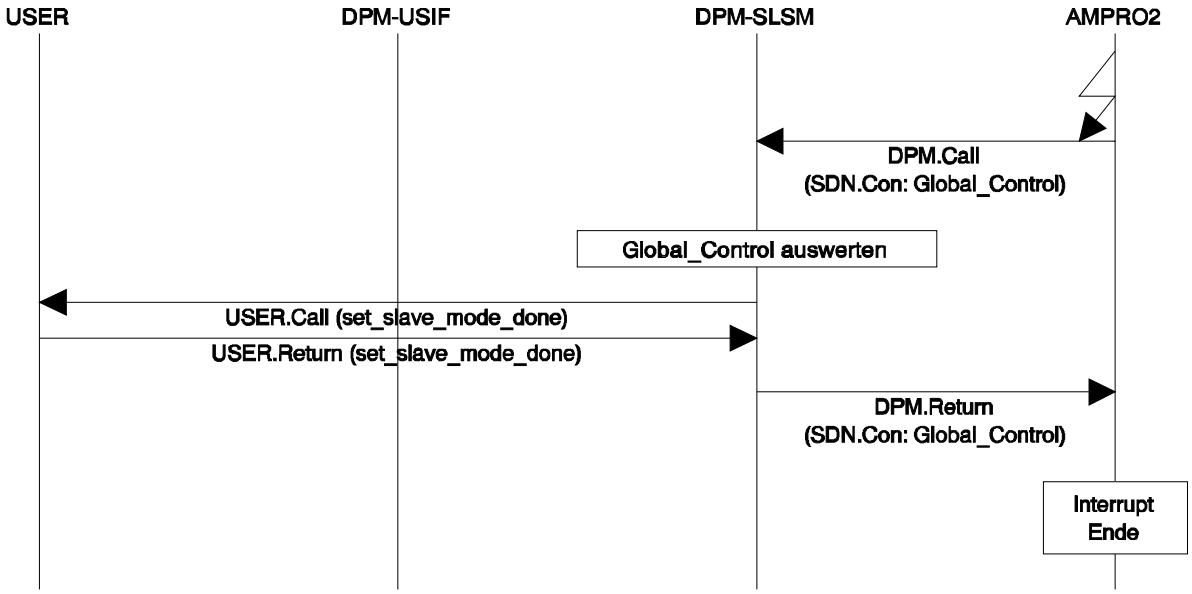
USER calls the function. Start a dummy job.



Evaluate confirmation of the dummy job and start the Global_Control service



Evaluate confirmation of the Global_Control service and inform USER

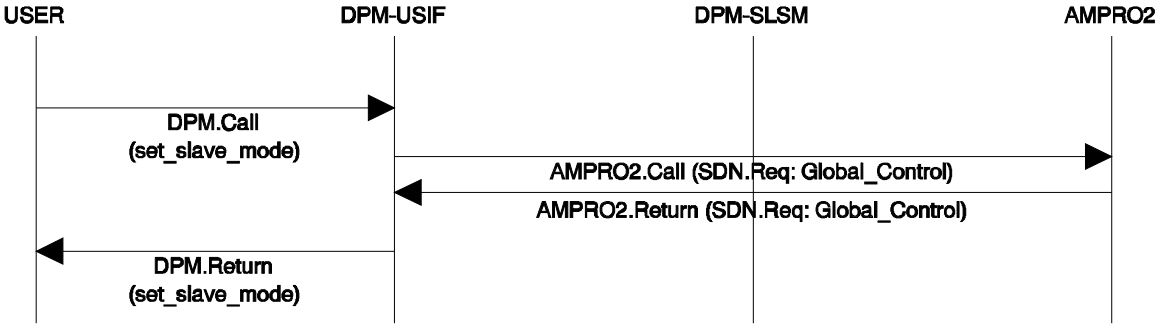


5.2.3.8.4.2 FREEZE/UNFREEZE Slave Operating Mode

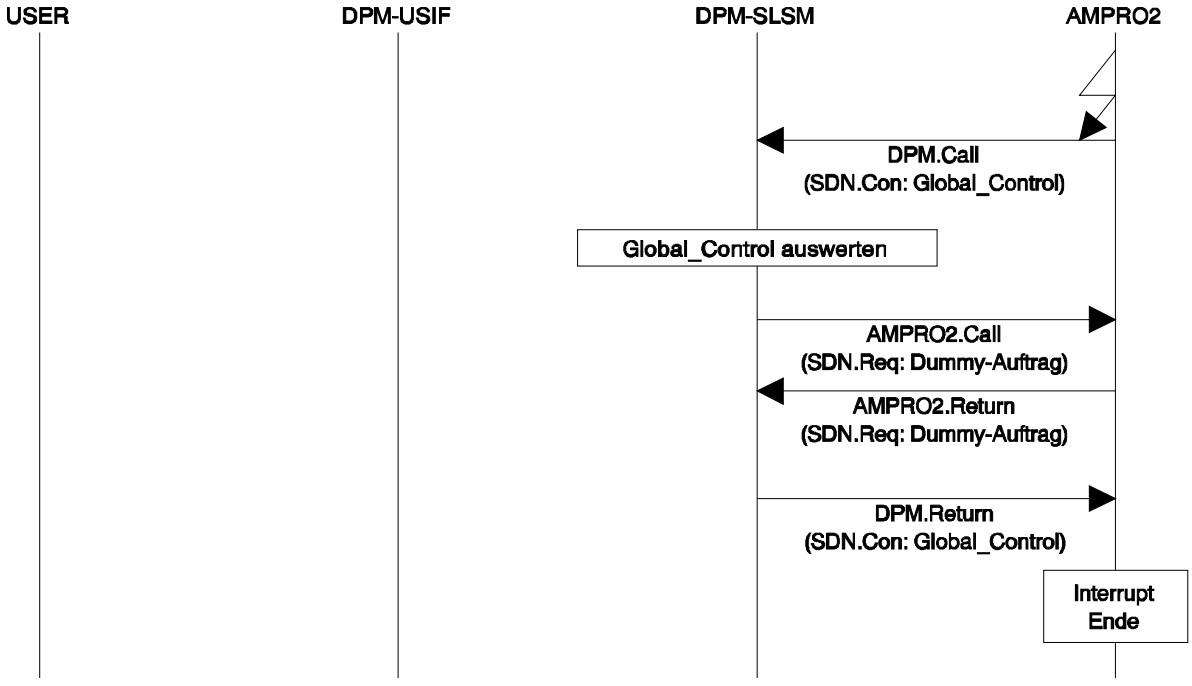
As soon as the USER calls the "set_slave_mode ()" function for FREEZE, the corresponding job is sent to AMPRO2. After the addressed slaves have received the indication, these slaves must freeze their inputs. Depending on the type of slave (i.e., purely hardware or purely software or mixed), the slave may not be able to react to the command immediately. New input data are still transferred even after the FREEZE command. AMPRO-DPM also takes this delay into consideration and maintains a wait time as for SYNC. Use of the "mark_cycle ()" command by the USER is not necessary. The USER does not receive the return message via the "set_slave_mode_done ()" CBF until this time has expired.
Also similar to the SYNC command, when the FREEZE command is used, the USER must suppress access to the input areas of the addressed slaves between the time the command is issued and the CBF is called.

Sequence charts:

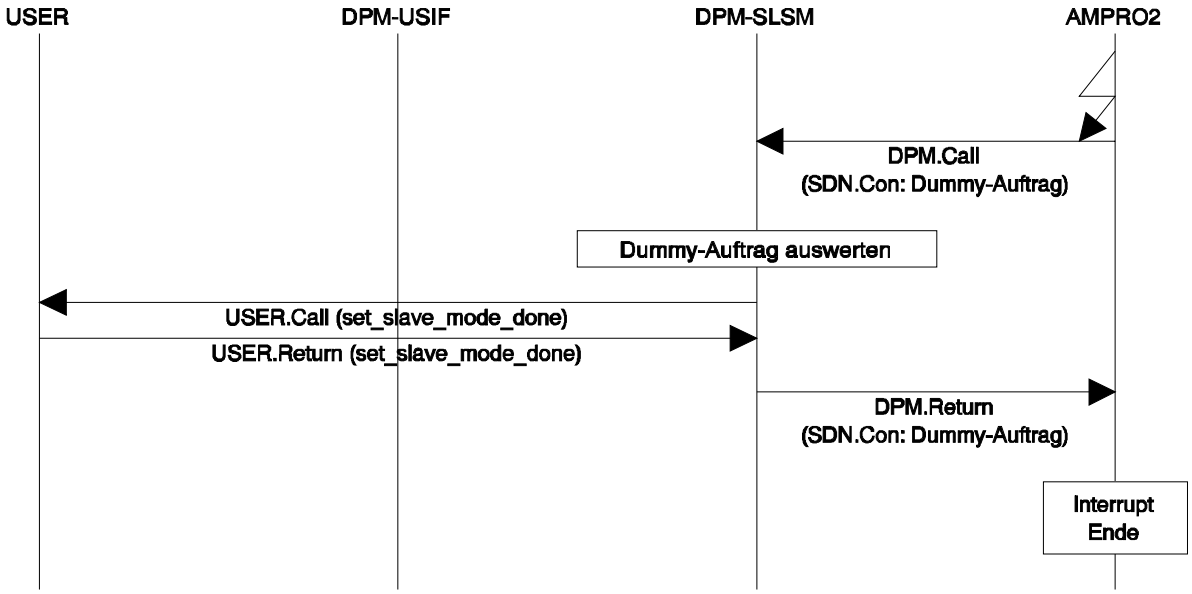
Function called by USER. Start Global_Control service.



Evaluate confirmation of the Global_Control service and start a dummy job



Evaluate confirmation of the dummy job and inform USER



5.2.3.8.4.3 Simultaneous Execution of SYNC or UNSYNC and FREEZE or UNFREEZE

When the USER specifies in the function call that SYNC or UNSYNC and FREEZE or UNFREEZE are to be executed simultaneously, the auxiliary jobs are omitted for both jobs. The FREEZE job is used instead of the dummy job for SYNC and vice versa.
As before, use of the "mark_cycle ()" command by the USER is not necessary since AMPRO-DPM maintains the two wait times. The USER must only ensure that neither the input nor output data areas can be accessed again until the "set_slave_mode_done ()" function is received.

5.2.3.9 set_slave_address (SSLACB_PTR)

5.2.3.9.1 Description

The "set_slave_address ()" command sets the PROFIBUS station address of a slave. This service can only be used for standard DP slaves.
Acceptance of the change in address by the slave is a prerequisite. This is not possible when the station address of the slave can only be changed with a DIP switch, for example. In addition, the USER must ensure that **neither** the current address of the slave (0_D-125_D or default address 126_D) **nor** the new address to be set is already set for another station of the bus system. The most reliable way to prevent address conflicts is to use a point-to-point link between the AMPRO-DPM module and the slave. The slave may not be located under the old address when in the SL_DATA state since it ignores all requests for address change when in this state. There are several ways for the USER to achieve this. When a slave is configured under the old station address, its processing may not be started with "add_slave ()" (see page 96 ff.) or it must be explicitly concluded with "withdraw_slave ()" (see page 100 ff.). Instead, the master can also be put into the MA_STOP state with "set_master_mode ()" (see page 104 ff.) since, although the master can send address change requests in this mode, the slaves are not processed.

To change the address of a slave, the USER transfers a correctly completed Set_slave_address control block (definition: SSLACB) to AMPRO-DPM. After the station address has been changed, the USER receives the return response together with the SSLACB via the "set_slave_address_done ()" CBF.

5.2.3.9.2 set_slave_address Control Block (SSLACB)

To specify the service in more detail, the user must transfer a pointer to the correctly completed set_slave_address control block (definition: SSLACB). This block is only required once for all slaves.

5.2.3.9.2.1 Header

Type	Designation	Direction of Access			
		USER		DPM	
		L	S	L	S
DPM_IFA_DATA_ATTR *	next_blk_ptr			X	X
DPM_IFA_DATA_ATTR *	prev_blk_ptr			X	X
Unsigned8	opcode (DPM_SET_SLAVE_ADDRESS)	X	X	X	
Unsigned8	subsystem (dpm_handle)	X	X	X	
L2_TYPE_ID_PTR	id_ptr (Can be used by the USER as desired)	X	X		

The header is filled out the same as the header description of the SLCB (see page 82 ff.) except that the function identifier DPM_SET_SLAVE_ADDRESS (see page 156) must be entered for *opcode* and the handle supplied by the "dpm_open ()" function (see page 86 ff.) must be entered for *subsystem*.

5.2.3.9.2.2 General Portion

Type	Designation	Direction of Access			
		USER		DPM	
		L	S	L	S
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to "set_slave_address_done ()" CBF	X	X	X	
Unsigned8	remote_add (0D-126D)	X	X	X	
Unsigned8	new_slave_add (0D-125D)	X	X	X	
Unsigned8	ident_no_high	X	X	X	
Unsigned8	ident_no_low	X	X	X	
Unsigned8	no_add_chg (DP_TRUE, DP_FALSE)	X	X	X	
Unsigned8	*Reserved*				
Unsigned8	status	X		X	X
Unsigned8	rem_slave_data_length	X	X	X	
DPM_PROC_DATA_ATTR *	rem_slave_data_ptr (pointer to an Unsigned8 array)	X	X	X	

Pointer to "set_slave_address_done ()" CBF:

This pointer must point to the "set_slave_address_done ()" CBF of the USER. See page 138.

remote_add:

The USER must enter the current address of the slave (0_D-126_D) here. This can also be the default address (126_D) as defined.

new_slave_add:

The USER must enter the address of the slave (0_D-125_D) to be set here. The default address (126_D) cannot be used here.

ident_no_high:
ident_no_low:

These two bytes are assigned the high or low byte of the PNO ident number of the slave. Each standard DP slave must be assigned an ident number by the PNO. See also the description of the DPMIB. AMPRO-DPM uses this number to determine whether the slave with the set *remote_add* is really the slave which the USER is thinking of.

no_add_chg:

This boolean parameter specifies whether the slave will or will not permit its address to be changed again after the current change in address. An entry of DP_TRUE permits later changes. An entry of DP_FALSE does not permit later changes.

status:

Since this variable provides the USER with information on the status of the service on the bus, it cannot be evaluated until after the block is returned with the "set_slave_address_done ()" CBF. See page 138.

rem_slave_data_length:

The USER can immediately transfer any data record to the slave with the change-of-address telegram. If this is to be done, the USER must fill a data block with these data and enter its length (1_D - 240_D bytes) in *rem_slave_data_length*. When no data exist, 0_D must be entered.

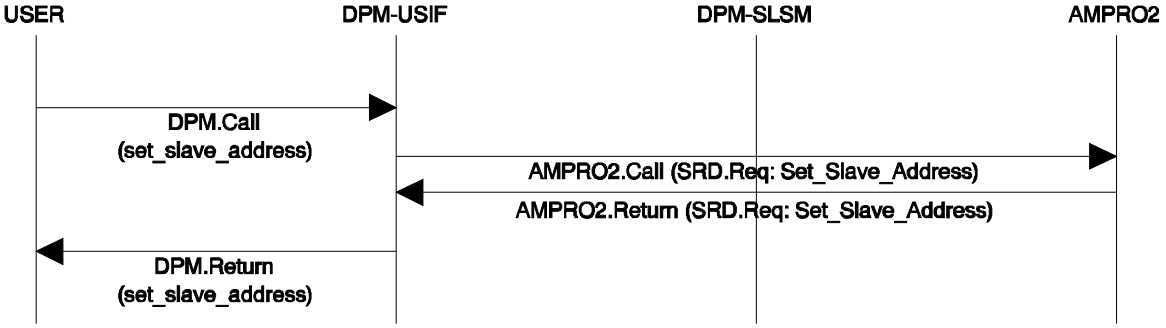
rem_slave_data_ptr:

When additional data are to be transferred, the pointer to these data (Unsigned8 array) must be entered here. When *rem_slave_data_length* is 0_D, this pointer can be disregarded.

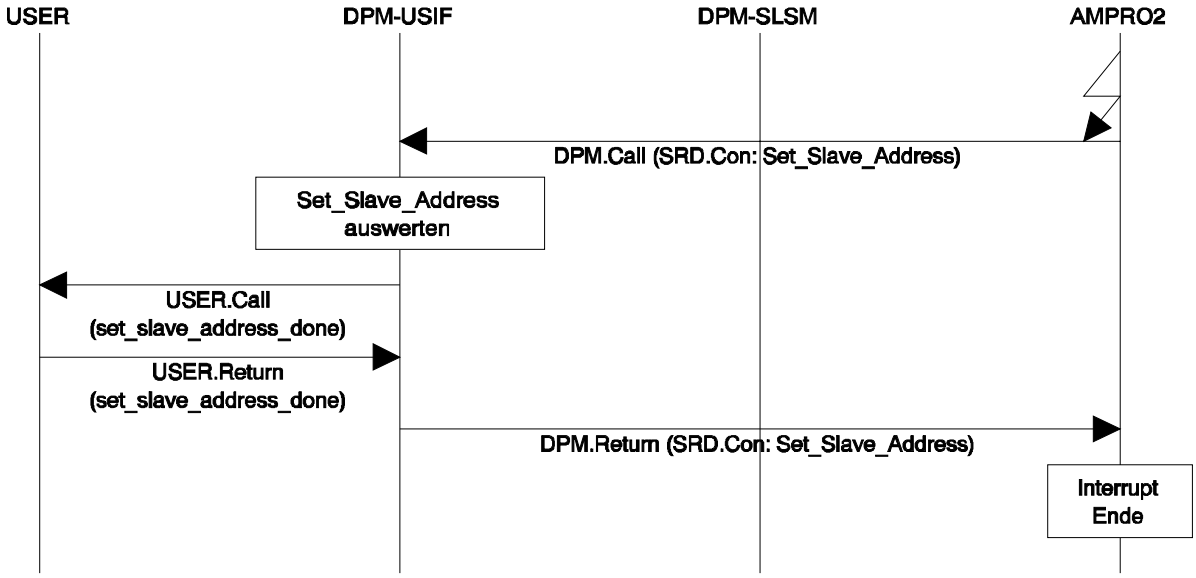
5.2.3.9.3 Call

Sequence charts:

Function is called by the USER and the Set_Slave_Address job is started.



Evaluate confirmation of the Set_Slave_Address job and inform USER



Sample call:

```
status = dpm_ptr -> set_slave_address ((SSLACB_PTR) &sslacb);
```

Possible return values for *status.* (This is the return value of the "set_slave_address ()" function and not the status in SSLACB.)

DPM_OK_CBF	The job was started correctly. Call of the CBF indicates end of execution.
ERR_DPM_REQ_ACTIVE	Error: The last "set_slave_address ()" or "init ()" call is still being processed.

5.2.3.10 mark_cycle (MARKCB_PTR)

5.2.3.10.1 Description

The "mark_cycle ()" service is used to synchronize the USER with AMPRO-DPM. The USER receives the return response to this command (i.e., the "mark_cycle_done ()" CBF - see page 139) when every activated slave has been addressed at least once since the function started. "mark_cycle ()" can only be called in the master modes MA_CLEAR and MA_OPERATE. If the master mode is changed while "mark_cycle ()" is being executed, the USER receives an appropriate error message in MARKCB with the CBF.
This service is primarily used to maintain the wait times before or after the SYNC/UNSYNC and FREEZE/UNFREEZE Global_Control services in accordance with the PROFIBUS-DP standard. Since these wait times are adhered to by AMPRO-DPM anyway (see use of the "set_slave_mode ()") service, if required at all, the USER can use the service for other purposes.

5.2.3.10.2 mark_cycle Control Block (MARKCB)

To specify the service in more detail, the user must transfer a pointer to the correctly completed mark_cycle control block (definition: MARKCB). This block is required globally only once.

5.2.3.10.2.1 Header

Type	Designation	Direction of Access			
		USER		DPM	
		L	S	L	S
DPM_IFA_DATA_ATTR *	next_blk_ptr			X	X
DPM_IFA_DATA_ATTR *	prev_blk_ptr			X	X
Unsigned8	opcode (DPM_MARK_CYCLE)	X	X	X	
Unsigned8	subsystem (dpm_handle)	X	X	X	
L2_TYPE_ID_PTR	id_ptr (can be used by the USER as desired)	X	X		

The header is completed the same as the header description of the SLCB (see page 82 ff.) except that the function identifier DPM_MARK_CYCLE (see page 156) must be entered for *opcode* and the handle (see page 86 ff.) supplied by the "dpm_open ()" function must be entered for *subsystem*.

5.2.3.10.2.2 General Portion

Type	Designation	Direction of Access			
		USER		DPM	
		L	S	L	S
DPM_CALL_BACK_ FUNC_ATTR *	Pointer to the "mark_cycle_done ()" CBF	X	X	X	
Unsigned8	status	X		X	X
Unsigned8	dia	X		X	X

Pointer to "mark_cycle_done ()" CBF:

This pointer must point to the "mark_cycle_done ()" CBF of the USER. See page 139.

status:

This entry is one of the return values for the USER. AMPRO-DPM enters the return values in accordance with the PROFIBUS-DP standard here. The USER cannot evaluate this entry until after the return of the block with the "mark_cycle_done ()" function. See page 139.

dia:

This entry is the second return value for the USER. AMPRO-DPM enters the return values in accordance with the PROFIBUS-DP standard here. The USER cannot evaluate this entry until after the return of the block with the "mark_cycle_done ()" function. See page 139.

5.2.3.10.3 Call

Sequence chart:

The "mark_cycle ()" service is executed the same as the "set_slave_mode ()" service except that the two jobs for "mark_cycle ()" are auxiliary jobs. See the sequence charts of the "set_slave_mode ()" service.

Sample call:

```
status = dpm_ptr -> mark_cycle ((MARKCB_PTR) &markcb);
```

Possible return values for *status*. (This is the return value of the "mark_cycle ()" function and not the status in MARKCB.)

DPM_OK_CBF	The job was started correctly. The call of CBF indicates end of execution.
ERR_DPM_REQ_ACTIVE	Error: The last "mark_cycle ()" or "init ()" call is still being processed.
ERR_DPM_INVALID_MODE	Error: Wrong master mode. Only possible in master mode MA_CLEAR or MA_OPERATE.

5.2.3.11 input_update (SLCB_PTR)

5.2.3.11.1 Description

Use of this function is only required for inputs in Buffered_Mode. The same also applies to "long" consistency since this is imaged in Buffered_Mode.
To receive current input data at all times, the user must request this with the "input_update ()" function. The pointer to the SLCB of the slave for which the inputs of the stated types are entered in the slave parameter record is used as the transfer parameter of the function. When this slave is in the DATA state (in any other state, the USER receives an error message) and AMPRO-DPM is in the master state OPERATE or CLEAR (in STOP status, the USER also receives an error message), AMPRO-DPM calls the "write_inp_data_to_pda ()" CBF. See page 141 ff.
First, the USER receives the pointer to the SLCB as the parameter of the CBF. The pointer to the process data area of the inputs of this slave is entered there (entry: *input_data_ptr*). Second, the USER receives a pointer-pointer to the buffer which has just been exchanged. The USER must then provide for the data transfer. There are two ways to accomplish this.

☞ If the process data area is shared memory area (i.e., dual-port RAM or similar), the USER must activate consistency control for the area specified by the process data pointer when inputs with "long" consistency are involved. The pointer-pointer is then used to copy all data from the buffer to the process data area. The SLCB contains the required length (entry: *input_data_len*). Consistency control is deactivated, and the CBF is concluded.
Inputs in Buffered_Mode are handled the same way except that activation and deactivation of the consistency control logic is omitted.

Caution:
Activation and deactivation of the consistency lock during the "write_inp_data_to_pda ()" CBF must never increase the run time for the CBF.

☞ When the data must be sent with an operating system message for example, the buffer can also be sent directly. In this case, the USER will have to supply a new buffer from AMPRO2 memory management which is then entered in the buffer pointer by the USER. At the end of the CBF, AMPRO-DPM expects a buffer with any contents to be valid. To make it easier to select the data buffer size, the buffer's number is entered in the SLCB (entry: *input_data_db_no*). This number can be used by the USER to select the appropriate "I2_mem_alloc_db**x** ()" function of AMPRO2 memory management.
When using this technique, the USER should not forget to transfer the length specification from the SLCB (entry: *input_data_len*) together with the buffer.

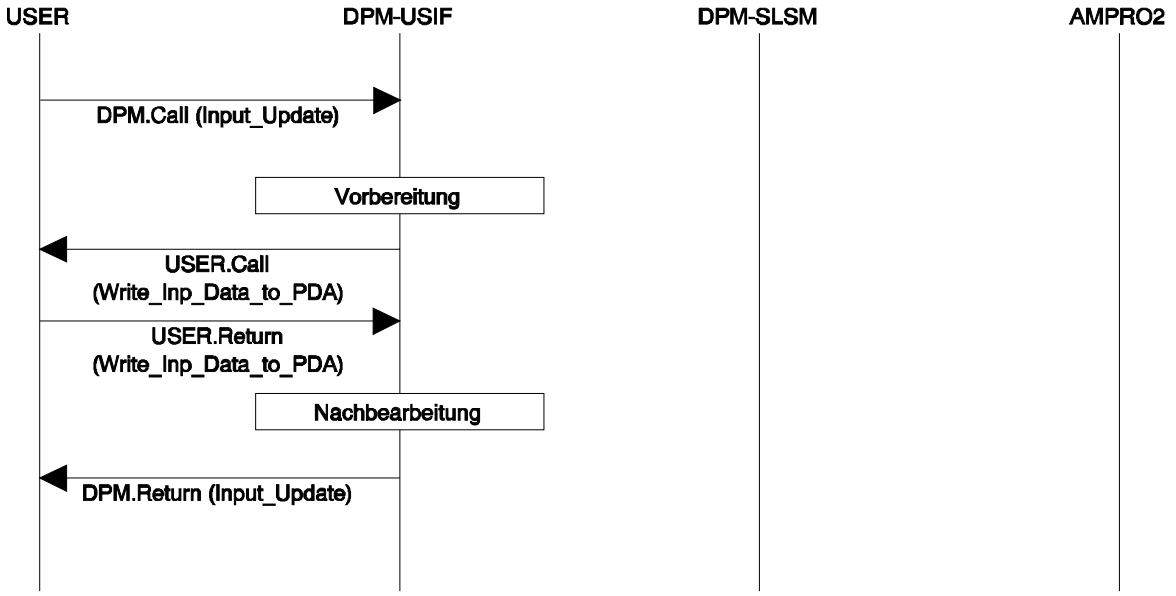
The "input_update ()" function must be called by the USER when necessary for a slave.
The frequency of several "input_update ()" calls in sequence should not exceed the activation interval of the "consistency_update ()" function. If "input_update ()" is called too frequently, an appropriate error message is generated for the USER and no new input data are available. Based on this error message, the USER can increase the call interval of the "input_update ()" function, for example.

The minimum time interval for updating the input data depends on the bus rotation time and the hardware base on which AMPRO-DPM is executed. Updating can usually be performed every several milliseconds (approx. 5 msec). Shorter update rates endanger the functionality of AMPRO-DPM because of the increased processor load. In addition, since new input data usually do not exist yet, the input data do not have to be updated anyway. The bus parameter T_{TR} also offers a time base one or more of which must be inserted between the "input_update ()" calls.

5.2.3.11.2 Call

Sequence chart:

"input_update ()" function called by the USER



Sample call: New input data are to be copied for slave no. x.

```
status = dpm_ptr -> input_update ((SLCB_PTR) &slcb[x]);
```

Possible return values for *status*:

DPM_OK		Service executed correctly
ERR_DPM_REQ_ACTIVE	Error:	The "init ()" call is still being processed.
ERR_DPM_NO_CONS_SLAVE	Error:	Neither "long consistency" nor Buffered_Mode is entered in the slave parameter record for the inputs of the slave.
ERR_DPM_NOT_ALLOWED	Error:	AMPRO-DPM is not in the OPERATE or CLEAR master state.
ERR_DPM_NOT_IN_DATA	Error:	The slave is not in DATA status. The data could not be updated.
ERR_DPM_CONS_REQ_ACTIVE	Error:	No new input data are available yet. The "consistency_update ()" function has not been called between the calls of "input_update ()".
ERR_DPM_NOT_PROCESSED	Error:	No new input data are available yet. The master has not yet polled the slave again.

5.2.3.12 output_update (SLCB_PTR)

5.2.3.12.1 Description

Use of this function is only required for outputs with "long" consistency. Buffered_Mode is not required for outputs.
To receive current output data at all times, the user must request these data with the "output_update ()" function. The pointer to the SLCB of the slave for which outputs with "long" consistency are entered in the slave parameter data record serves as the transfer parameter of the function. When this slave is in the DATA state (in any other state, the USER receives an error message), AMPRO-DPM requests the current output data of the slave from the USER. The "read_outp_data_from_pda ()" CBF is called. See page 141 ff. This CBF is always started within the "output_update ()" function. See sequence chart. The USER receives a pointer to the SLCB as a parameter of the CBF. Located there is the pointer to the process data area of the outputs of this slave (entry: *output_data_ptr*). The USER also receives a pointer-pointer to an empty buffer. The USER must now provide for the data transfer. There are two ways to accomplish this.

- ☞ If the process data area is a shared memory area (i.e., dual-port RAM or similar), the USER must first activate consistency control for the area specified by the process data pointer. All data are then copied from the process data area to the buffer via the pointer-pointer. The SLCB contains the required length (entry: *output_data_len*). Consistency control is then deactivated, and the CBF is concluded.

  Caution:
  Activation and deactivation of the consistency block during the "read_outp_data_from_pda ()" CBF must never increase the run time of the CBF.

- ☞ When the USER has received the output data with an operating system message for example, the buffer can also be sent directly to AMPRO-DPM. In this case, the USER must first receive the empty buffer and then add the buffer with the new ouputs to the buffer pointer-pointer. The empty buffer can continue to be used or it can be returned with the "I2_mem_free_db ()" function of AMPRO2 memory management.
  When using this technique, the USER should not forget to transfer the length specification from the SLCB (entry: *output_data_len*) together with the buffer.

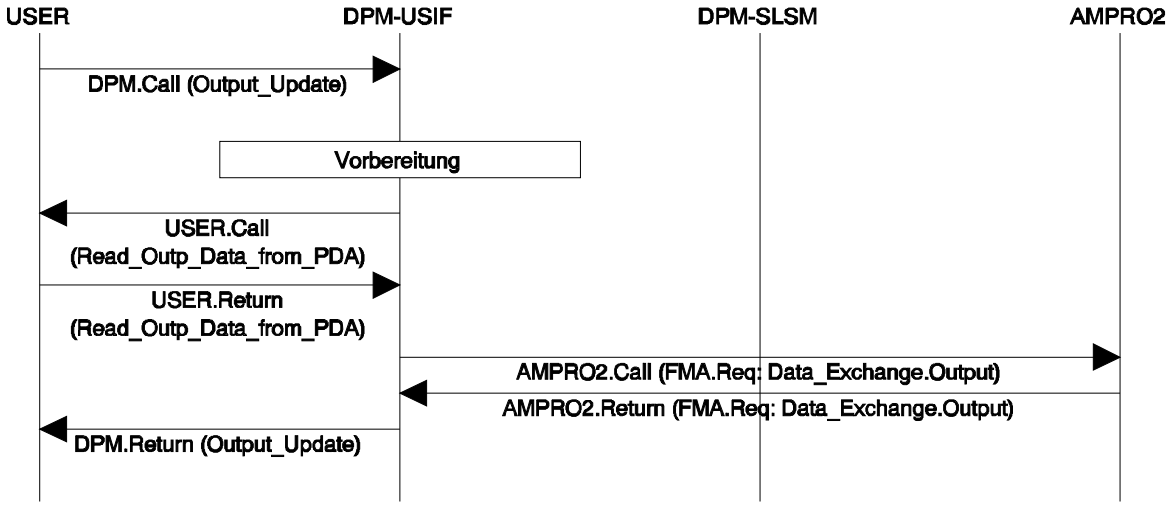
When AMPRO-DPM has received a buffer with new ouputs from the USER, it transfers this buffer to AMPRO2 with the MAC_REPEAT_EXCHANGE service for further use in the data cycle. In response, AMPRO-DPM receives an empty buffer which is used for the next "output_update ()" call.
If the slave is not included in the data cycle while the "output_update ()" function is being executed, the data buffer which has just been transferred is accepted anyway. This data buffer is then sent immediately after the slave returns to the data cycle. This means that the USER can "preset" the data buffer. When the USER does not preset the data buffer, the last valid data buffer is sent when a slave returns to the DATA state. However, these data may be out-of-date for the application process. A special_function bit (i.e., LCCO) has been defined to circumvent this behavior. When this special function is activated, the current output buffer is internally cleared when necessary by AMPRO-DPM.

The "output_update ()" function is usually called by the USER when needed (e.g., after an access to the input data area).
When this function must also be called cyclically, the same restrictions apply as with the "input_update ()" function. Call frequency should not be greater than the current target rotation time. If greater, the USER receives an appropriate error message. A maximum frequency of not less than 5 msec must be selected.

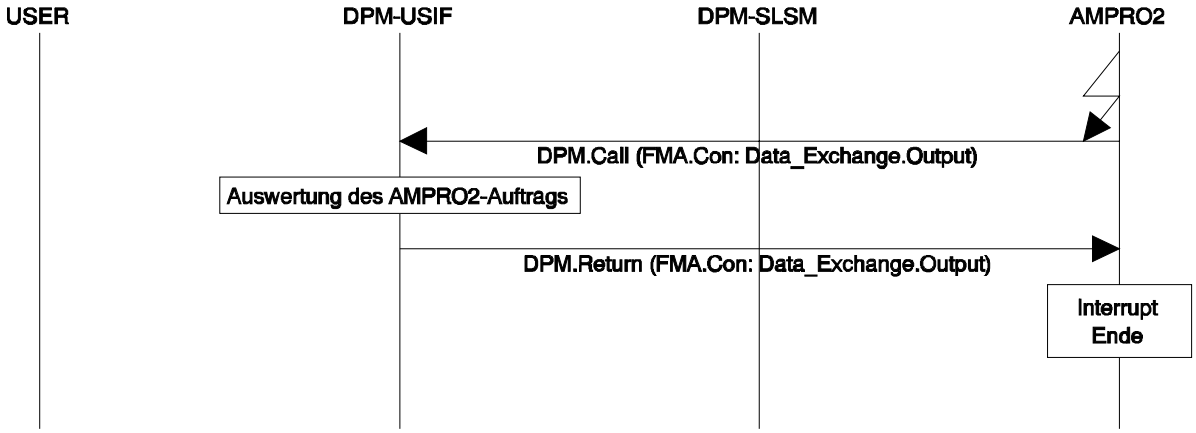
5.2.3.12.2 Call

Sequence chart:

The "output_update ()" function is called, and the buffer exchange is started by the USER.



Evaluation of the AMPRO2 request and post-processing



Sample call: New output data are to be transferred for slave no. x.

```
status = dpm_ptr -> output_update ((SLCB_PTR) &slcb[x]);
```

Possible return values for *status*:

DPM_OK		Service executed correctly
ERR_DPM_REQ_ACTIVE	Error:	The "init ()" call is still being processed. The output data were not accepted.
ERR_DPM_NO_CONS_SLAVE	Error:	Neither "long" consistency nor Buffered_Mode is entered in the slave parameter record for the outputs of the slave.
ERR_DPM_NOT_ALLOWED	Error:	AMPRO-DPM is not in OPERATE or CLEAR master status. The output data were not accepted.
ERR_DPM_NOT_IN_DATA	Error:	The slave is not in DATA status but the data were accepted.

ERR_DPM_CONS_REQ_ACTIVE	Error:	The current output data buffer was accepted, and the data will be sent later. An "output_update ()" which was triggered shortly before is still active. The call frequency of "output_update ()" is too high. DP bus performance is too low.
ERR_DPM_NOT_PROCESSED	Error:	The last output data could not be sent and have been overwritten by new output data. The call frequency of "output_update ()" is too high. DP bus performance is too low.
ERR_DPM_BUFFER_OVERWRITE	Error:	The last output data could not be sent and have been overwritten by new output data. The call frequency of "output_update ()" is too high. The function has been activated several times.

5.2.3.13 consistency_update (void)

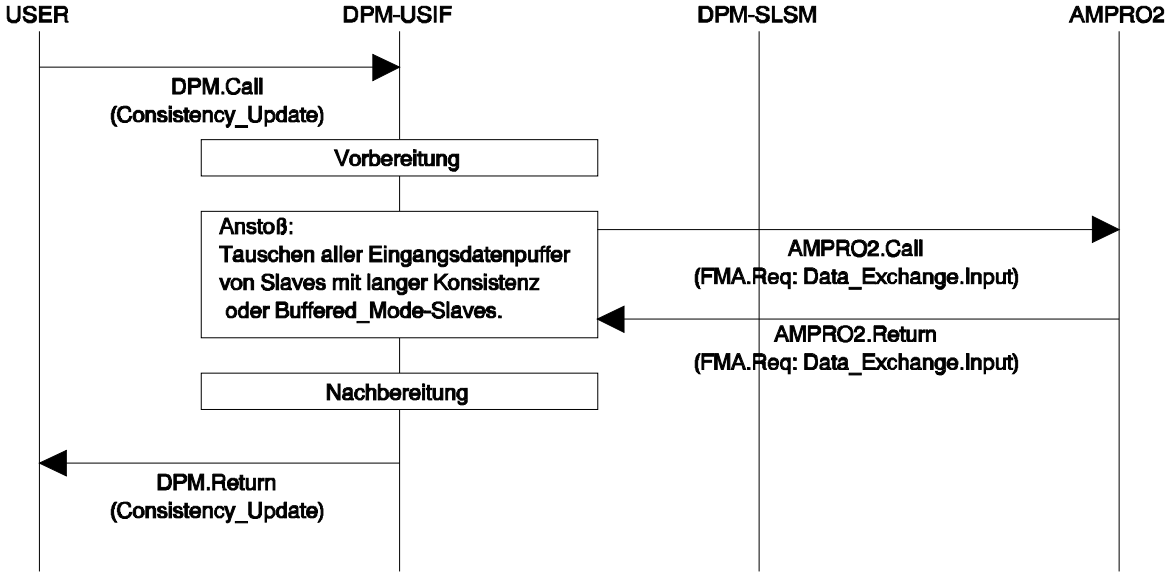
5.2.3.13.1 Description

Use of "long" consistency and Buffered_Mode requires that cyclic updating of the inputs within AMPRO-DPM be exited. This is the only way to provide the USER with a current input buffer via the synchronous "input_update ()" service. See page 118 ff.
The "consistency_update ()" function executes a complete update cycle for the inputs. The "consistency_update_done ()" call back function (i.e., CBF) also exists for this function. This CBF marks the end of a complete "consistency_update ()" call.
The USER must call "consistency_update ()" cyclically. We recommend starting the cyclic call of this function before "input_update ()" is called for the first time. A good time to start the cyclic call is after initialization of AMPRO-DPM or before master mode changes from STOP to CLEAR. To ensure new input data are actually available after each "consistency_update ()", the update interval should not be shorter than the target rotation time or not shorter than 10 msec (processor load).
An error message is generated when the call is performed too frequently.

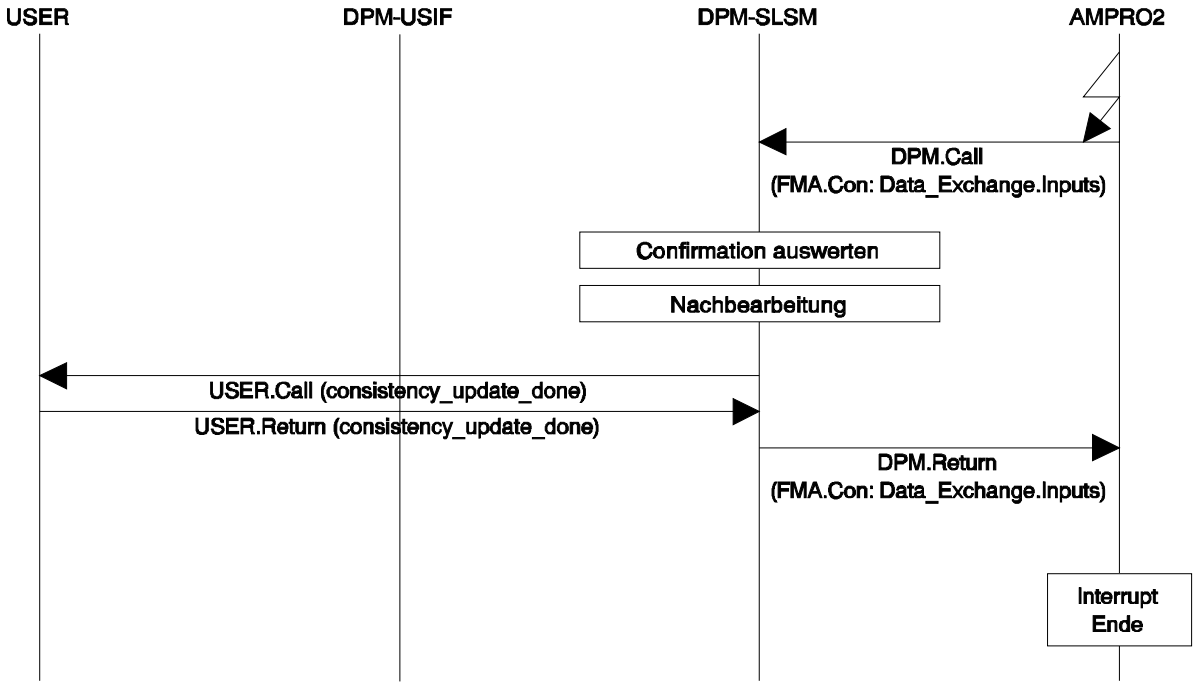
5.2.3.13.2 Call

Sequence chart:

Call the "consistency_update ()" function, and start buffer exchange within AMPRO-DPM



Evaluate confirmation of the "consistency_update ()" job, and inform USER after all exchange jobs have been returned



Sample call: The inputs are to be updated for slaves with "long" consistency or slaves in Buffered_Mode.

```
status = dpm_ptr -> consistency_update ();
```

Possible return values for *status*:

DPM_OK_CBF		Service executed correctly. The "consistency_update_done()" CBF is called.
ERR_DPM_REQ_ACTIVE	Error:	The "init ()" call is still being processed.
ERR_DPM_CONS_REQ_ACTIVE	Error:	Function is already active. The "consistency_update_done()" CBF is not called.
ERR_DPM_NO_CONS_SLAVE	Error:	No consistent slave or no slave in Buffered_Mode exists. The "consistency_update_done()" CBF is not called.
ERR_DPM_NOT_ALLOWED	Error:	AMPRO-DPM is not in OPERATE or CLEAR master status. The "consistency_update_done()" CBF is not called.

5.2.3.14 timer_expired (void)

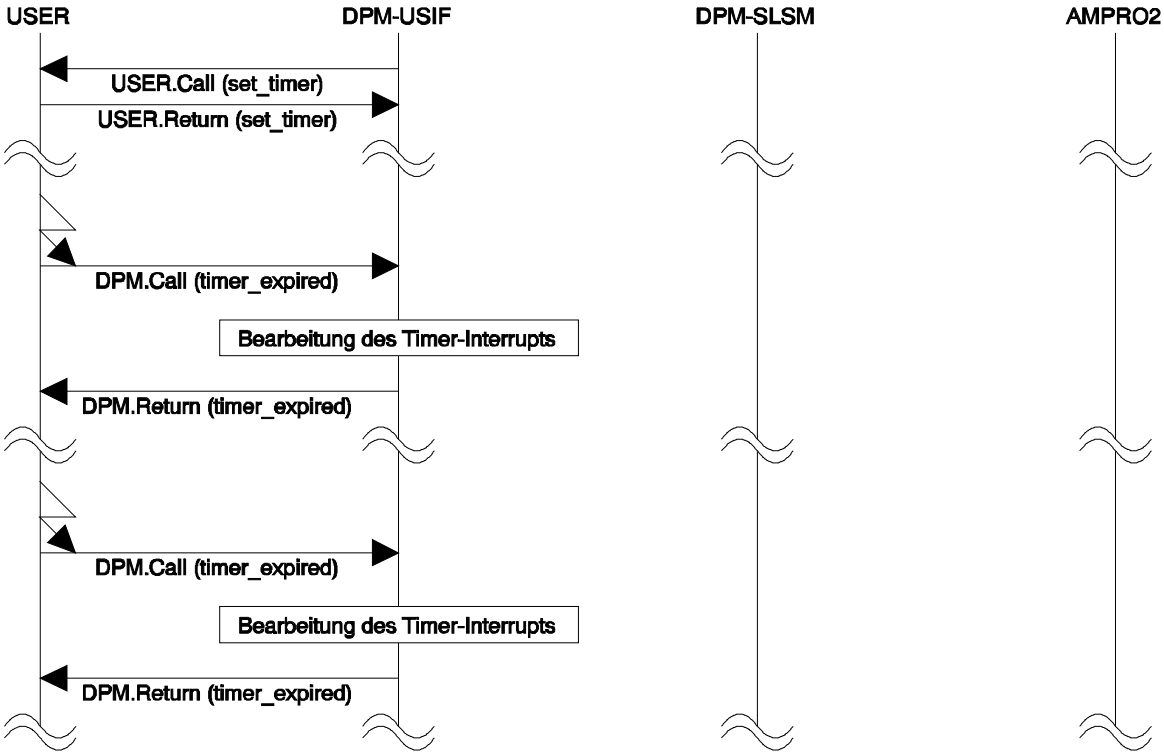
5.2.3.14.1 Description

Some internal AMPRO-DPM functions (e.g., monitoring of the Min_Slave_Interval or the Dx_Control_Interval) require time monitoring. The USER must provide AMPRO-DPM with a timer since the run time, setting accuracy precision and read-access technique of this timer are dependent on the hardware. AMPRO-DPM initiates the start of this timer with the "set_timer ()" CBF. See page 140 ff. The user receives a double word (Unsigned32) with the timer starting value in milliseconds as the transfer parameter of this function. The USER must start the timer in this function with the parameterized value and then conclude the

function. The size of the starting value depends on the timer values for the Min_Slave_Interval and the Dx_Control_Interval which the USER transferred with the "init ()" command. See page 88 ff. The starting value is usually half the value of the Dx_Control_Interval. A greater value is generated internally by AMPRO-DPM when the smaller value expires a number of times.
As soon as the timer has expired, the USER must call the "timer_expired ()" AMPRO-DPM function and restart the timer with the value transferred at the beginning. The cycle of the timer function call must continue until AMPRO-DPM has transferred a new time parameter to the USER with another call of "set_timer ()". The USER must terminate the current timer, load the timer with the new value and then start the timer again. When AMPRO-DPM transfers the value 0_H, the USER must only stop the timer. During a USER call of "timer_expired ()", the "timer_expired ()" function may not be called again until the first call has been concluded.
If AMPRO-DPM recognizes that the bus cannot be processed temporarily during the "timer_expired ()" function, the USER receives a different return value than the normal one. If this happens several times, AMPRO-DPM reports this error to the USER by calling the "bus_accessible ()" CBF. See page 140 ff. This CBF is already called during the call of the "timer_expired ()" function.

5.2.3.14.2 Call

Sequence chart:



Sample call:

```
status = dpm_ptr -> timer_expired ();
```

Possible return values for *status*:

DPM_OK	Service transferred to AMPRO2 correctly
DPM_OK_CBF	Service transferred to AMPRO2 correctly but the bus cannot be accessed at this time. If this condition persists, the "bus_accessible ()" CBF is started after "timer_expired ()" has been called several times. See page 140 ff.
ERR_DPM_REQ_ACTIVE	Error: The last "timer_expired ()" or "init ()" call is still being processed.

5.2.3.15 close (Unsigned8)

5.2.3.15.1 Description

The opposite of the "dpm_open ()" function (see page 86 ff.) is required to stop AMPRO-DPM in a defined state. This function is called "close ()". As the name indicates, unlike "dpm_open ()", this function is not separately defined. Instead the USER receives a pointer (among others) to "close ()" after "dpm_open ()" is called in the AMPRO-DPM call structure (see page 80 ff.).
Before "close ()" can be called during running operation, all slaves must be deactivated with "withdraw_slave ()". See page 100 ff. All other functions of AMPRO-DPM and all CBF calls must be concluded. In addition, the master must be in MA_STOP status. "close ()" can be called when all these preparations have been concluded. The handle supplied by "dpm_open ()" is used as the transfer parameter. After the call, AMPRO-DPM releases all remaining resources of AMPRO2 memory management and clears all internal management structures. The USER can now rearrange AMPRO2 memory management, prepare his system for a new start or a restart, or continue with another job. The USER must report again with "dpm_open ()" before the AMPRO-DPM functions can be used again after "close ()".

5.2.3.15.2 Call

Sample call:

Function definition from the "\COMMON\DPM_COMM.H" file

```
extern  DPM_PTR DPM_IFA_FUNC_ATTR    dpm_open   (Unsigned8 DPM_IFA_DATA_ATTR *);
```

Required variables:

```
SLCB            DPM_USER_DATA_ATTR   slcb[MAX_ANZ_SLAVES];
DPM_PTR         dpm_ptr;
Unsigned8       dpm_handle;
Unsigned16      status;
```

Function portions:

```
/* Call the OPEN function */
dpm_ptr = dpm_open ((Unsigned8 DPM_IFA_DATA_ATTR *) &dpm_handle);

     ...
     ...  /* Use the AMPRO-DPM functions */
     ...

/* Call the CLOSE function */
status = dpm_ptr -> close (dpm_handle);
```

Possible return values for *status*:

DPM_OK		Complete conclusion of AMPRO-DPM tasks was executed correctly.
ERR_DPM_WRONG_HANDLE	Error:	The handle transferred as the parameter is not permitted.
ERR_DPM_SLAVE_ACTIVE	Error:	At least one slave activated before with "add_slave ()" has still not been deactivated with "withdraw_slave ()".
ERR_DPM_REQ_ACTIVE	Error:	At least one previously started service has still not been completely concluded.
ERR_DPM_INVALID_MODE	Error:	The master is not in MA_STOP status.

5.2.3.16 dpm_l2_cb_server (L2_APB_PTR)

5.2.3.16.1 Description

Starting with version V2.0, AMPRO2 offers the capability of reporting back all jobs via call back functions (CBF). Each of these jobs is transferred to AMPRO2 with an application block (APB) and reported back with this APB. The *subsystem* entry of each APB must contain the number of the function which is to be called as CBF.
In the ASIC interrupt routine which the USER must provide, this number is used to select and call the appropriate CBF. Since a subdistributor (i.e., "dpm_l2_cb_server ()") handles this distribution for AMPRO-DPM, the USER must not know all the internal CBFs between AMPRO-DPM and AMPRO2. The USER must only know the range of CBF numbers to which AMPRO-DPM is assigned. When the USER receives an APB with a CBF number from this range in the interrupt routine (i.e., the main distributor), the USER calls the "dpm_l2_cb_server ()" function and specifies the APB.
The **CBF numbers 0_D to 69_D** are permanently reserved for AMPRO-DPM.

5.2.3.16.2 Call

Sample call:

An example of the interrupt routine of the USER is shown below. It is based on the example from the AMPRO2 application notes. Since portions of functions for interrupt processing may still have to be added before and after this function, this example only concerns itself with the CBF distributor.
While the interrupt routine is being processed, *dpm_ptr* which was completed by the "dpm_open ()" function (see page 86 ff.) must be globally known. The "dpm_l2_cb_server ()" function could have been entered during the initialization of the USER system in a pointer known to the interrupt routine instead. The contents of this pointer would then have been called during the interrupt routine instead of the above-stated call.

```
void L2_CALL_BACK_CODE_ATTR    ampro2_cbf_distributor    (void)
{
     L2_APB_PTR      apb_ptr;


     l2_aspc2_int_handler();

     while      ( apb_ptr = l2_con_ind() )
     {
          if    (apb_ptr->subsystem <= 69)
          {
               dpm_ptr->dpm_l2_cb_server (apb_ptr);
          }
          else
          {
               /* Here the CBF-distributors for other components may be added */
               /* or the "error()"-function may be called in case of a wrong  */
```

```
                /* subsystem-value. */
            }
        }
}
```

5.2.4 Call Back Functions (CBFs) of the USER

As already stated in the section on the communication model on page 61, CBFs are used to inform the USER of the status of a previously started AMPRO-DPM function, certain slave states or errors which have occurred.
CBFs are functions without return values which must be supplied by the USER. See section on the call structure for the USER starting on page 81. Their call roughly corresponds to the occurrence of an interrupt. Although the effect of these functions is irrelevant to AMPRO-DPM, their run times should be kept as short as possible. As transfer parameter, the USER usually first receives a job block directly (the "error ()" function - see description starting on page 143) or a pointer to this block (i.e., other functions). When the "state_report ()" function is involved (see description starting on page 129), the USER may only read-access the transferred block while the function is running. For all other functions with block pointers, the USER receives the block back. Transfer of the status messages from AMPRO-DPM to the USER is performed, if necessary, with the other parameter (type Unsigned16). This status word corresponds to the return value of normal functions.

The CBFs are divided into functions which originated when the USER triggered a service and functions which are triggered by AMPRO-DPM and are not dependent on USER requests. All CBFs which are called after USER requests are identified with a "..._done" in the function name.

Caution:

All CBFs which are called because of a USER job cannot only be called at the end of the user function but also any time beforehand. AMPRO-DPM starts the corresponding CBF as soon as the job has been executed. Depending on the priority of the USER job and the duration of execution, this point in time can be either before or after the end of the USER request. The USER must always be prepared for the possibility that the CBF for a job concluded with DPM_OK_CBF may already have been called.
In actual practice, this requirement can be implemented with two busy flags (i.e., one for the function and one for the CBF). Before each function is called, the USER sets both flags to busy. Both flags are reset after the function has been concluded with DPM_OK. After the DPM_OK_CBF return value of the function, only the function flag is cleared. In the CBF, the other flag is reset. In both cases, the current function call has been concluded when both flags have been cleared again. If the USER system does not have a multi-tasking or multi-processing environment, the BUSY function flag can be omitted.

5.2.4.1 init_done (Unsigned16)

When the "init ()" function started by the USER (see page 88 ff.) has been executed, the "init_done ()" function is called as the return message for the user. AMPRO-DPM receives the pointer to this function from the USER in DPMIB together with the job. The transfer parameter status word contains information on the status of the job. At the end of the function, the job block (i.e., DPMIB) can be used again by the USER.
Very different delay times can occur between the time when the "init ()" function is called and the time when the CBF is called. This depends on the physical setup of the bus and the parameterization. If, for example, the USER has transferred a master station number in the DPMIB which is larger than the HSA of the active PROFIBUS station already on the bus, the master will never receive the token and never obtain sending rights. This suppresses the CBF call permanently. The same situation also occurs when the bus lines are short circuited. Even during normal operation without such errors, long delays can occur before the master receives the token. Assume, for example, the lowest baud rate (i.e., 9.6 kbaud) and parameterization of all time values to their maximum values, and, in addition, poor distribution (i.e., very high and very low numbers) of the station numbers of the active stations with a heavy bus load. Under these conditions, it might take several **days** (days is not a typographical error) until the master receives the token. This example shows how important realistic bus parameters, adjustment of the bus load to the actual requirements and a correct (optimized if possible) bus setup are.

After the "dpm_open ()" function has been called, the USER receives a pointer to the AMPRO-DPM call structure but, to prevent operator errors, this structure only contains a few function pointers at this time. See the "dpm_open ()" function starting on page 86. All other pointers have the value ZERO. The remaining pointers are not entered in the structure until the "init_done ()" function is called.

The "init_done ()" function is also one of those CBFs which can already be called while the "init ()" function is still running. This can occur particularly when a bus with a high baud rate is being used without any additional masters or active stations. The USER must then use internal flags to ensure that the "bus is ready for operation" information is not lost. The "init ()" function itself is always (when no errors occur) concluded with the DPM_OK_CBF return message even when the CBF was already called before the end of the function. See also the introduction to this section.

Possible values for the status word:

DPM_OK	Service executed correctly

5.2.4.2 state_report (SLCB_PTR)

The "state_report ()" function is used to inform the USER of the previous, next or current status of the slave. The transferred SLCB may only be read-accessed by the USER and only while the function is being executed. After the function has been concluded, AMPRO-DPM regains sole access rights to the block.

5.2.4.2.1 Slave States with Parameterizing Master

The following states are defined for each slave. See section on coding rules for slave states on page 155.

Operating Mode	***Explanation***
DEACT	The slave is not being processed.
DIAG1	The slave is being polled for the first time or has experienced a failure. The slave now reports diagnoses until it can be parameterized.
PRM	The slave is being parameterized.
CFG	The slave is being configured.
DIAG2	The slave reports diagnoses or long-term status messages until it assumes another status due to the messages.
DIAG2_STATUS	The slave is signaling a status message.
S7_GET_CFG	The configuration will be fetched from the slave next. Only applicable to S7_GET_CFG slaves.
DATA	The master is exchanging data with the slave.
DATA_NA	The slave malfunctioned after the master exchanged data with the slave.
PRM_UNLOCK	The slave is being deactivated.
STOPPED	The slave is not being processed since its handler was stopped due to AUTOSTOP master operating mode.

DEACT is the original status of each slave before "add_slave ()". The DEACT state can be entered in the SLCB by the USER as *actual_state* before the first "add_slave ()" command. However, this is not absolutely necessary and is only used to improve comprehension and simplify internal communication for the USER. When the "state_report ()" is called for the first time, DEACT is always returned as *last_state*. After the "withdraw_slave_done ()" CBF is called, DEACT is the *actual_state* of the slave.
The "state_report ()" function is called each time the operating mode changes. In addition, in the DIAG2 state, every new diagnostic telegram is reported with "state_report ()".

AMPRO-DPM usually uses a special feature of AMPRO2 (i.e., the repeat jobs) in connection with the ASPC2 PROFIBUS ASIC. A repeat job permits AMPRO-DPM to specify continuation of the slave state

machine expected with the current state of the slave when issuing jobs. If the expected process occurs, the ASIC can continue processing alone without any help from AMPRO-DPM. AMPRO-DPM does not interfere until a change in the slave state occurs. This is the only way to reduce the extreme load of the processor despite the ASIC at a maximum baud rate of 12 Mbaud to a realistic value. For the USER, this means that only *changes* in state are usually reported with "state_report ()".
Repeat jobs usually cannot be used while the slave handler is booting since a change in slave status is usually required after each job from AMPRO-DPM to AMPRO2. The ASIC usually does not begin processing the AMPRO-DPM jobs alone until the first *DATA → DATA* transition occurs. After this transition occurs, at least one data telegram has been transferred to the slave. Cyclic processing of the slave can now begin, and an initial current image of the inputs of the slave is available to the master.

In addition to the slave states described in the DP standard, the DIAG2_STATUS, S7_GET_CFG, DATA_NA and STOPPED states reported by AMPRO-DPM have been added.
The USER could generate the DIAG2_STATUS and DATA_NA states but he would have to obtain this information from several locations which would take time. Since AMPRO-DPM has this information at its direct disposal, the use of these states can simplify processing for the USER considerably.
The slave state S7_GET_CFG only occurs with type S7_GET_CFG S7 slaves. See "copy_s7_get_cfg_data ()" function on page 141. STOPPED is a special state which only occurs in connection with the AUTOSTOP master operating mode.

When a slave is in the DATA state, it can always provide status messages to the master. However, these states are transferred in the DIAG2 state the same as "real" error messages. Determination of whether the diagnostic message of the current DIAG2 state is an error message or a status message is based on the transition sequence of the states (i.e., *DATA → DIAG2 → DATA* (status message) or *DATA → DIAG2 → not DATA* (error message)). The DIAG2_STATUS state has been provided to relieve the USER of this time-consuming evaluation.
When a slave in the DATA state requests a diagnostic telegram, its next status is DIAG2_STATUS. After the master has received the diagnoses, it changes the slave state back to DATA if a status message is involved. The corresponding next state (e.g., DIAG2 for another diagnostic request of the slave) is entered for all other messages. A transition from *DIAG2_STATUS → DIAG2_STATUS* is not possible. This state can only be assumed once. The slave can only assume DIAG2_STATUS from the DATA state.

DATA_NA is similar to the DIAG1 state. The slave no longer responds to the requests of the master. Despite this, the slave is still supplied with current data telegrams. When the slave becomes available again after a period of time (any length of time), a complete new registration does not absolutely have to be performed after DATA_NA. If the slave is able to process the received data telegram correctly, its next state is DATA. Such transitions usually occur when the threshold monitoring time of the slave is disabled or the time has not yet expired. If the slave does not return to the bus until after the threshold monitoring time has expired, it reports an error for the data telegram and is included again via DIAG1.

The following table shows the transitions in operating mode which are reported. The two right-hand columns specify which entry is made for the status change in the master diagnostic field for this slave. See the "init ()" function. DTL stands for Data_Transfer_List. SD stands for System_Diagnostic. After initialization, the value 0_B is entered for both areas. This means that the corresponding slave is in the DEACT state. However, the DTL is not transferred with the current status. Instead, it is cleared after one half of one Dx_Control_interval by the DPM-USIF and transferred to the master diagnostic area as specified by the PROFIBUS-DP standard. The DPM-SLSM only sets the bits when necessary but does not clear them. Changes of the DTL are thus reported with a delay of up to one half of one Dx_Control_interval. The SD list contains the current state.

Operating Mode		*Information for the USER*	*DTL*	*SD*
Previous	***Current***			
DEACT	DIAG1	The slave will be included next in the processing cycle with DIAG1 telegrams.	0B	1B
STOPPED	DIAG1	A slave which was previously removed from the processing cycle due to the AUTOSTOP master mode will be re-included next in the processing cycle with DIAG1 telegrams.	0B	1B
DIAG1	PRM	The slave will be parameterized next.	0B	1B
PRM	CFG	The slave will be configured next.	0B	1B
CFG	DIAG2	The slave will be diagnosed next.	0B	1B
PRM	DIAG2	Only applicable to Siemens DP slaves with SPM ASIC: The slave will be diagnosed next.	0B	1B
DIAG2	DIAG2	The last diagnostic message of the slave was entered in the data area specified by SLCB. The diagnostic status remains valid.	0B	1B
DIAG2	S7_GET_CFG	Only applicable to S7_GET_CFG slaves: The configuration will be requested next from the slave.	0B	1B
DIAG2	DATA	The last diagnostic message of the slave was entered in the data area specified by the SLCB. Data communication with the slave can be started next.	1B	0B
S7_GET_CFG	DATA	Only applicable to S7_GET_CFG slaves: The current configuration has been successfully fetched. Data communication with the slave can be started next.	0B	1B
DATA	DATA	Data communication with the slave is taking place. This transition is usually reported once per startup of the slave handler.	1B	0B
DATA	DATA_NA	The slave failed after data communication with it took place.	0B	1B
DATA_NA	DATA	After a short failure, the slave can be processed again with data telegrams without having to be added again.	1B	0B
DATA, DATA_NA	DIAG2_STATUS	The slave will be diagnosed next. The slave has just received a data telegram, however.	1B	1B
DIAG2_STATUS	DATA	The diagnosis which was requested last was a status message. The slave is receiving data telegrams again.	1B	0B
DIAG2_STATUS	DIAG2	The diagnosis which was requested last was an error message. The slave continues to receive diagnostic telegrams.	0B	1B
Any mode except PRM_UNLOCK, DEACT or STOPPED	DIAG1	The slave has failed permanently or has reported a serious error. Transition from DIAG2_STATUS or DATA is not possible in AUTOSTOP master mode.	0B	1B
Any mode except DIAG1, DEACT or STOPPED	PRM_UNLOCK	The slave is being removed from the processing cycle.	0B	1B
DIAG1, PRM_UNLOCK, STOPPED	DEACT	The slave was deactivated due to a USER request. Transition from STOPPED can only take place in AUTOSTOP master mode.	0B	0B
DIAG1, PRM_UNLOCK	STOPPED	The slave was removed from the processing cycle due to AUTOSTOP master mode.	0B	0B

5.2.4.2.2 Slave Handler with Parameterizing Master

5.2.4.2.2.1 Transitions in Slave State during Normal Operation

The following figure shows the states and transitions in state for a slave in accordance with the PROFIBUS-DP standard.
The STOPPED state cannot be assumed during normal master operation. After PRM_UNLOCK a branch is always made to DEACT since this state can only be assumed after a "withdraw_slave ()" call. The two shaded states (i.e., DIAG2_STATUS and DATA) are the states in which data communication is performed with the slave.

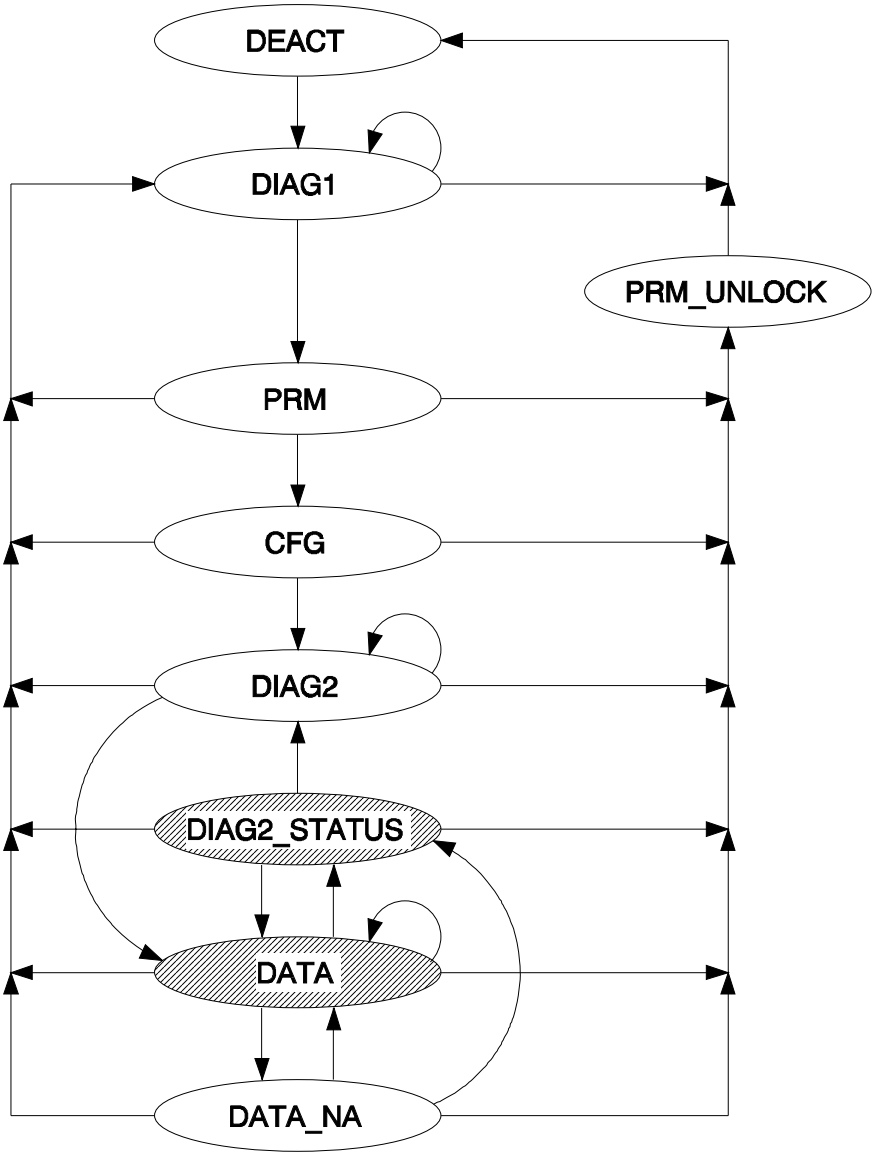


Figure 6: Transitions in slave state with parameterizing master during normal operation

5.2.4.2.2.2 Transitions in Slave State during AUTOSTOP Master Mode

In AUTOSTOP master mode, transitions from DATA and DIAG2_STATUS to DIAG1 are not possible since a PRM_UNLOCK sequence is always started from these two states when an error occurs. The DATA_NA state does not occur at all. Transition from PRM_UNLOCK or DIAG1 to DEACT is made when the slave was deactivated before with "withdraw_slave ()". Transition to STOPPED occurs immediately after the DATA or DIAG2_STATUS state is exited. The two shaded states (i.e., DIAG2_STATUS and DATA) are the states in which data communication with the slave takes place.

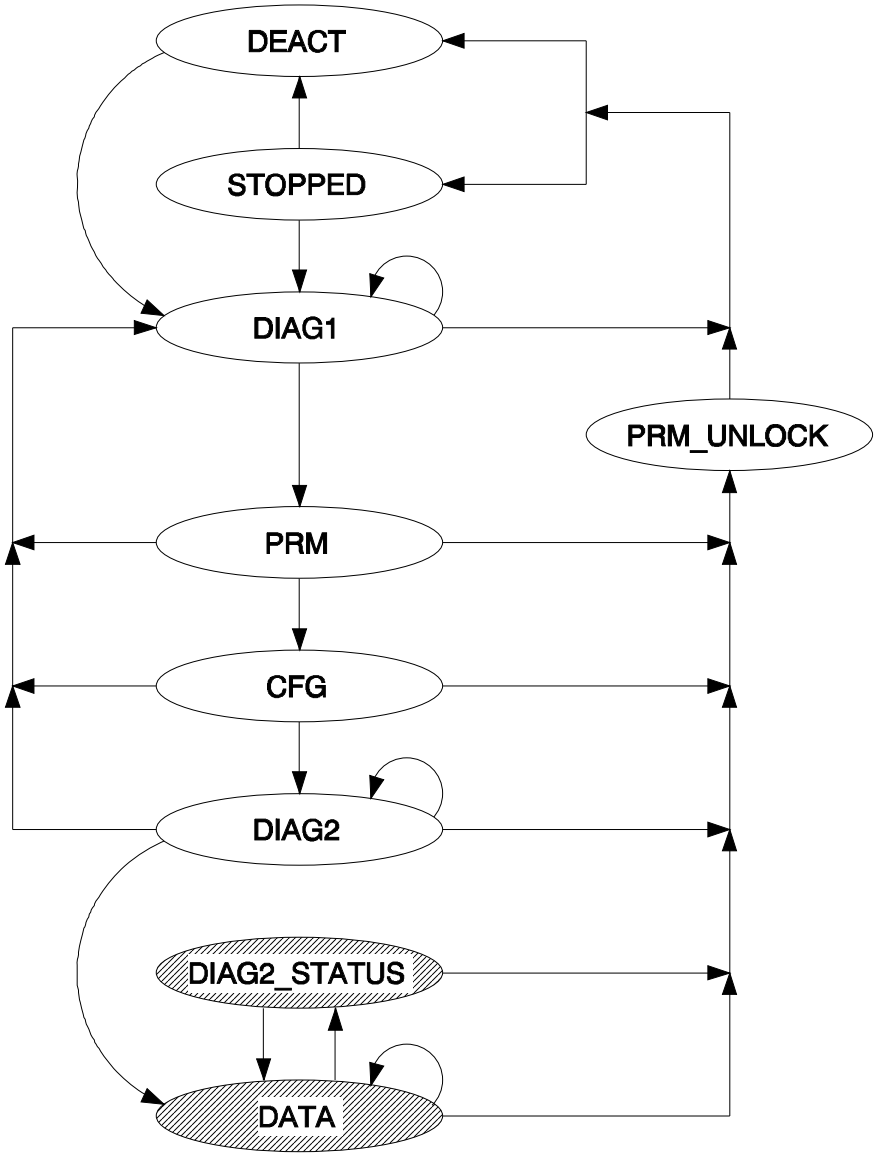


Figure 7: Transitions in slave state with parameterizing master during AUTOSTOP mode

5.2.4.2.2.3 Transitions in Slave State during S7_GET_CFG Slave Mode

Transitions in slave state during S7_GET_CFG slave mode are the same as those of a standard DP slave. An additional S7_GET_CFG slave state must be integrated in the slave handler for operation of S7_GET_CFG slaves. See page 129 ff.

The S7_GET_CFG state is achieved when the slave reports in the DIAG2 state that it is ready for data communication. Instead of changing immediately to the data cycle, the current configuration is requested from the slave and then provided to the USER. See page 141 ff. The change to the data cycle occurs after the configuration has been requested.

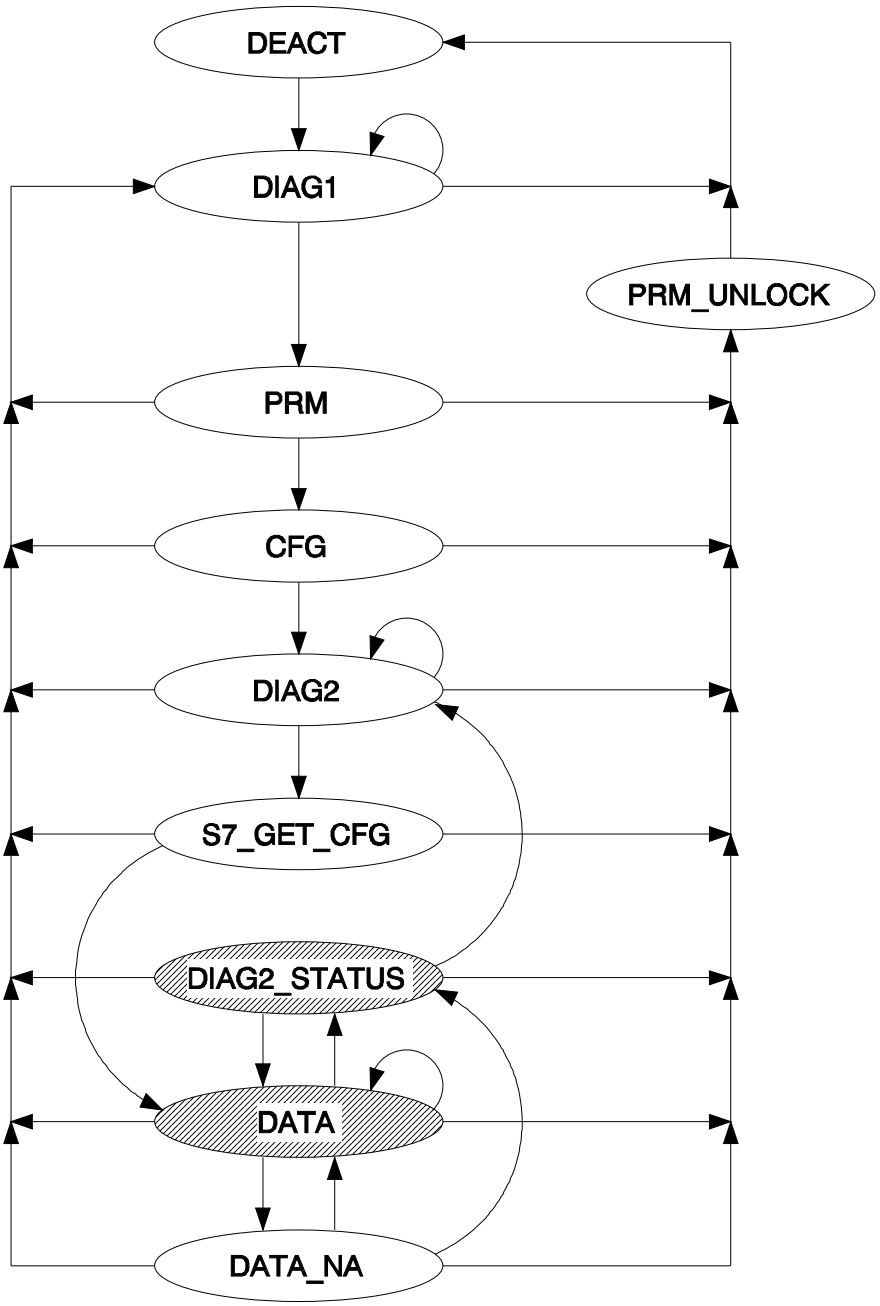


Figure 8: Transitions in slave state with parameterizing master during S7_GET_CFG mode

Slave States with Shared_Input Master
The following operating modes are defined for each Shared_Input slave.

Operating Mode	*Explanation*
DEACT	The slave does not react because it is turned off, for example.
DIAG1	The slave reports diagnoses until it is parameterized by another master.
SIO_GET_CFG	The configuration of the slave is being checked.
SIO_DIAG2	The slave reports diagnoses or status messages until it assumes another state due to the messages.
SIO_RD_IO	The master reads the inputs of the slave.
STOPPED	The slave is not processed since its handler has been stopped by the AUTOSTOP master mode.

DEACT, DIAG1 and STOPPED correspond to the states for the parameterizing master. The slave exits the DIAG1 state as soon as another master (or the parameterizing master of the slave) has parameterized the slave. Based on the configuration of the slave, the Shared_Input master checks to determine whether it is communicating with the desired slave. The Shared_Input master then reads the diagnoses in the SIO_DIAG2 state until the slave reports that it is ready for data communication (or until an error occurs which causes a transition back to DIAG1). After transition to SIO_RD_IO, no more diagnoses are read from the slave until the slave or the parameterizing master is stopped or fails. The slave state usually changes to DIAG1 afterwards. In AUTOSTOP master mode, the new slave state is STOPPED. Every change in state and every new diagnosis is reported to the USER with "state_report ()".
The following table shows the transitions in state which are reported. The same conditions apply to DTL and SD as for the parameterizing master.

Operating Mode		*Information for the USER*	*DTL*	*SD*
Previous	*Current*			
DEACT	DIAG1	The slave will be included next in the processing cycle with DIAG1 telegrams.	0B	1B
STOPPED	DIAG1	A slave which was previously removed from the processing cycle because of AUTOSTOP master mode will be included next in the processing cycle with DIAG1 telegrams.	0B	1B
DIAG1	SIO_GET_CFG	The configuration of the slave will be checked next.	0B	1B
SIO_GET_CFG	SIO_DIAG2	The slave will be diagnosed next.	0B	1B
SIO_DIAG2	SIO_DIAG2	The last diagnostic message of the slave was entered in the data area specified in SLCB. The diagnostic state remains valid.	0B	1B
SIO_DIAG2	SIO_RD_IO	The last diagnostic message of the slave was entered in the data area specified in SLCB. The inputs of the slave can be read next.	1B	0B
SIO_RD_IO	SIO_RD_IO	The inputs of the slave are read. This transition is usually reported once per startup of the slave handler.	1B	0B
Any mode except DEACT	STOPPED	The slave was removed from the processing cycle due to AUTOSTOP master mode.	0B	1B
SIO_GET_CFG, SIO_DIAG2, SIO_RD_IO	DIAG1	The slave has failed or has reported a serious error (not from SIO_RD_IO in AUTOSTOP master mode).	0B	1B
Any mode	DEACT	The slave is deactivated.	0B	0B

The primary difference between the slave diagnosis of the Shared_Input master and the slave diagnosis of the parameterizing master is contained in status byte 1. When the parameterizing master is involved, bit 7 (i.e., "Master_Lock") must be cleared before the slave assumes the DATA state since otherwise another master has already parameterized the slave. When the Shared_Input master is involved, "Master_Lock" must be set before the slave assumes the SIO_RD_IO state since otherwise the slave has not yet been parameterized by another master.

5.2.4.2.3 Slave Handler with Shared_Input Master

5.2.4.2.3.1 Transitions in Slave State during Normal Operation

The following figure shows several other transitions in state for the Shared_Input master in addition to the transitions in operating state of the slave with the parameterizing master. The shaded SIO_RD_IO state is the only state in which the inputs are requested cyclically by the slave.

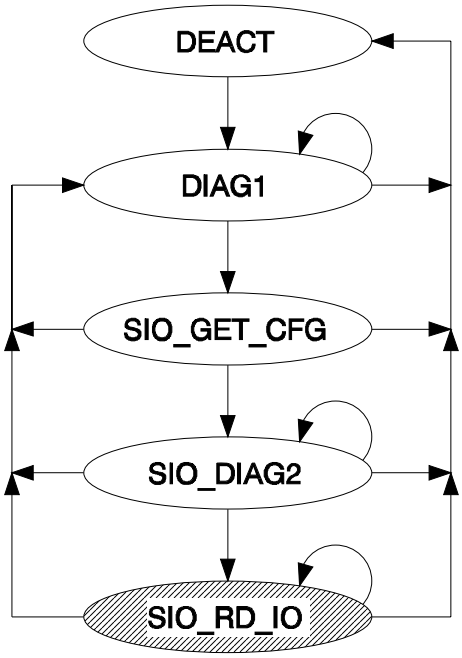


Figure 9: Transitions in slave state with the Shared_Input master during normal operation

5.2.4.2.3.2 Transitions in Slave State during AUTOSTOP Master Mode

In AUTOSTOP master mode, the transition from SIO_RD_IO to STOPPED occurs when the slave or the parameterizing master fails. Transition to DEACT does not occur until after the "withdraw_slave ()" function is called. The shaded SIO_RD_IO state is the only state in which inputs are requested cyclically by the slave.

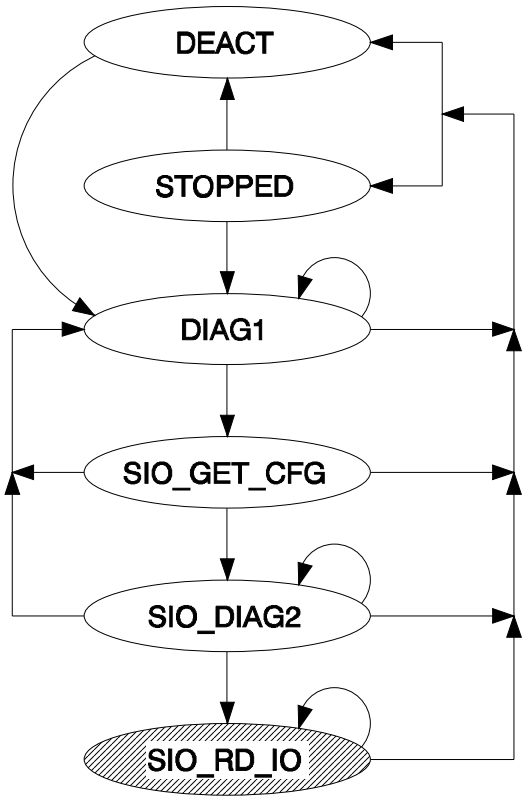


Figure 10: Transitions in slave state with the Shared_Input master during AUTOSTOP mode

5.2.4.3 withdraw_slave_done (SLCB_PTR, Unsigned16)

When called and processed correctly, this function informs the USER that the slave which was deactivated with the previous "withdraw_slave ()" call (see page 100 ff.) has been removed from the processing cycle in a defined state and the memory areas for the slave have been returned to AMPRO2. The SLCB of this slave can now be completed again by the USER. The second transfer parameter (i.e., the status word) contains information on the job.

Possible values for the status word:

DPM_OK	Service executed correctly

5.2.4.4 set_master_mode_done (SMMCB_PTR, Unsigned16)

After the "set_master_mode ()" function started by the USER has been executed, the "set_master_mode_done ()" function is called as the return message for the user. AMPRO-DPM receives from the USER the pointer to this function in SMMCB together with the job. The second transfer parameter (i.e., the status word) contains information on the job. After the function, the job block (SMMCB) can be used again by the USER.

Possible values for the status word:

DPM_OK	Service executed correctly

5.2.4.5 set_slave_mode_done (SSMCB_PTR, Unsigned16)

After the "set_slave_mode ()" function (see page 107 ff.) started by the USER has been executed, the "set_slave_mode_done ()" function is called as the return message for the user. AMPRO-DPM receives from the USER the pointer to this function in SSMCB together with the job. The second transfer parameter (i.e., the status word) contains information on the job. After the function, the transferred job block (i.e., SSMCB) can be used again by the USER. When necessary, there can be eight of these functions present if they are being used to separate the return message of the jobs.

Possible values for the status word:

DPM_OK	Service executed correctly

5.2.4.6 set_slave_address_done (SSLACB_PTR, Unsigned16)

After the "set_slave_address ()" function (see page 113 ff.) started by the USER was executed, the "set_slave_address_done ()" function is called as the return message for the user. AMPRO-DPM receives from the USER the pointer to this function in SSLACB together with the job. The second transfer parameter (i.e., the status word) contains information on the job. In addition to the information in the status word, further error classifications can be made via the *status* parameter in SSLACB. The entries there correspond to the AMPRO2 or PROFIBUS error messages. After the function, the job block (i.e., SSLACB) can be used again by the USER.

Possible values for the status word:

DPM_OK	Service executed correctly
ERR_DPM_L2_RES_SSLA	Error: An error occurred while the job was being processed. The *status* parameter in SSLACB provides information on the type of error.

Possible values for the *status* SSLACB parameter when the status word has the value DPM_OK:

DPM_STATUS_OK	Service executed correctly

Possible values for the *status* SSLACB parameter when the status word has the value ERR_DPM_L2_RES_SSLA:

L2_STATUS_UE	Error:	The slave is not designed to process the service.
L2_STATUS_RR	Error:	The slave cannot process the service due to a (temporary) lack of resources.
L2_STATUS_RS	Error:	The service is not activated on the slave (at this time).
L2_STATUS_WD	Error:	The service was terminated due to a short circuit on the bus.
L2_STATUS_NA_TIMEOUT	Error:	The slave does not respond.
L2_STATUS_NA_BUFFER_ERROR	Error:	The slave responds with a wrong telegram length in the response telegram.
L2_STATUS_NA_DOUBLE_TOKEN	Error:	A bus failure has occurred.
DPM_STATUS_RE	Error:	The slave responds incorrectly.

5.2.4.7 mark_cycle_done (MARKCB_PTR, Unsigned16)

When this function is called in response to start the "mark_cycle ()" USER function (see page 117 ff.), the master has completed a full polling cycle of the slaves. From this time on, it is ensured that all slaves have received current data. The second transfer parameter (i.e., the status word) contains information on the job. In addition to the status word, further state and error classifications can be made with the *status* and *dia* parameters in MARKCB. After the function, the job block (i.e., MARKCB) can be used again by the USER.

Possible values for the status word:

DPM_OK	Service executed correctly

Possible values for the *status* MARKCB parameter:

DPM_STATUS_OK	Service executed correctly
DPM_STATUS_NO	While "mark_cycle ()" was being executed, the USER changed the master mode to MA_STOP causing the service to be terminated.

Possible values for the *dia* MARKCB parameter:
(This parameter is not valid unless the *status* MARKCB parameter has the value DPM_STATUS_OK.)

DP_FALSE	No diagnosis available. All activated slaves were included in the data cycle while "mark_cycle ()" was being processed.
DP_TRUE	Diagnosis available. At least one activated slave was not included in the data cycle while "mark_cycle ()" was being processed.

5.2.4.8 set_timer (Unsigned32)

Each time this CBF is called, the user must stop (if already started) and restart the timer provided for AMPRO-DPM.
The transfer parameter specifies the new run time of the timer in milliseconds. When the transferred value is 0_H, the timer must only be stopped. AMPRO-DPM can call a new "set_timer ()" at any time except when the function is already being called. See section on communication model starting on page 61.
If, for any reason, the USER is unable to start the timer, he must conclude the "set_timer ()" call and then conclude DP communication via "set_master_mode ()" (the concluding state of the master must be MA_STOP) since the slaves cannot be processed correctly without the timer. Slave processing may not be started again until the timer is available.
The "set_timer ()" CBF is called for the first time after the master mode changes from MA_STOP to MA_CLEAR. See "set_master_mode ()" function starting on page 104. This CBF is always called within the call of the "set_master_mode ()" USER function. From this point on, the USER must call the "timer_expired ()" function cyclically as described starting on page 124. As a rule, "set_timer ()" is only started again once (i.e., to deactivate the timer when the master mode changes from MA_CLEAR to MA_STOP). This CBF is always called before the "set_master_mode_done ()" CBF is called. See page 138.

With this AMPRO-DPM version, the parameters of the timer (e.g., accuracy, fluctuation and drift) can be specified by the USER based on user capabilities and requirements. Changes in timer accuracy only affect the DX_Control_Intervall. All other times are directly maintained by the ASPC2 PROFIBUS ASIC with the accuracy called for by PROFIBUS. However, fluctuations in excess of 10% should be avoided so that the update interval for the Data_Transfer list does not vary too much, for example. See section on the DPMIB starting on page 88. When later firmware versions are involved or when another PROFIBUS ASIC is used later, the timer must still be used for other tasks when necessary. For this reason, we strongly recommend that an extremely precise timer be used not only with regard to the interface to the USER but also for later expansions.

5.2.4.9 bus_accessible (Boolean)

This is one of the CBFs which is called by AMPRO-DPM without a previous job from the USER. The parameter value DP_FALSE from "bus_accessible ()" informs the USER that a bus malfunction preventing further communication with the slaves (e.g., a bus short circuit) has occurred during running operation. See also description of the "init_done ()" CBF on page 128. All management and job data are retained during this time, and the operating state of the master is "frozen.". As soon as the bus can be accessed again after correction of the error (e.g., correction of the short circuit), the USER is informed of this with a new call of the CBF. This time the parameter value is DP_TRUE. As a rule, the slaves must be re-included by AMPRO-DPM in the processing cycle via the DIAG1 state.
When required, this CBF is always called within the "timer_expired ()" USER function. See page 124 ff.

Possible values for the status word:

DP_FALSE	Bus cannot be accessed at this time.
DP_TRUE	Bus can be accessed again.

In addition to the procedure described here, the USER can also generate his own mechanism for recognizing the occurrence of a short circuit. AMPRO2 offers the new MAC_REQ_WITHDRAW service for this purpose. After this service is called, all jobs which are still chained are returned to their senders with a separate error identifier (i.e., L2_STATUS_WD). AMPRO-DPM handles this identifier the same as the L2_STATUS_NA_TIMEOUT return message. When the USER utilizes this service, he can disregard the information supplied by the "bus_accessible ()" CBF and enter a dummy function for this CBF instead. Or, the USER can stop calling the function when the function pointer has a value of ZERO.

*5.2.4.10 write_inp_data_to_pda (SLCB_PTR, Unsigned8 L2_DATA_ATTR * DPM_IFA_DATA_ATTR *)*

AMPRO-DPM only calls the "write_inp_data_to_pda ()" CBF when the USER has previously started the "input_update ()" function (see page 118 ff.) for a slave with inputs with "long" consistency or with inputs in Buffered_Mode. The description of the "input_update ()" function also contains a comprehensive description of this CBF. A word for status messages is not required for this function.

*5.2.4.11 read_outp_data_from_pda (SLCB_PTR, Unsigned8 L2_DATA_ATTR * DPM_IFA_DATA_ATTR *)*

The "read_outp_data_from_pda ()" function is only called by AMPRO-DPM when the USER has previously started the "output_update ()" function (see page 121 ff.) for a slave with outputs with "long" consistency. The description of the "output_update ()" function also contains a comprehensive description of this CBF. A word for status messages is not required for this function.

*5.2.4.12 copy_s7_get_cfg_data (SLCB_PTR, Unsigned8 L2_DATA_ATTR * DPM_IFA_DATA_ATTR*, Unsigned8)*

AMPRO-DPM only calls this function when the slave is an S7_GET_CFG slave. This type of slave can contain both a physical and a logical configuration. The physical configuration is available when the slave starts up. After parameterization and configuration, the S7 slave determines its logical configuration. The logical configuration is available when the slave is ready for data communication after the DIAG2 phase. The SLCM does not branch directly to the data cycle at this point. Instead, it requests the logical configuration of the S7 slave. The "copy_s7_get_cfg_data ()" CBF is then called with this configuration, and the SLSM branches to the data transfer phase.

The USER receives the pointer to the SLCB as a parameter of the CBF. The USER also receives a pointer-pointer to a buffer with the current configuration data. The USER must provide for a check of the configuration. There are two ways to accomplish this.

- If the configuration check is not time-consuming, evaluation can be performed with the configuration supplied with the pointer-pointer. The third transfer parameter of "copy_s7_get_cfg_data ()" (i.e., the Unsigned8 value) contains the current length of the configuration data.

- If the configuration check is time-consuming, the USER can add a new AMPRO2 data buffer to the pointer-pointer and perform the evaluation during the cyclic portion of his application.
  In this case, the USER must obtain a new buffer from AMPRO2 memory management and enter this buffer in the pointer-pointer. AMPRO-DPM expects a buffer with any content at the end of the CBF to be valid. The buffer's number is entered in the SLCB (entry: *max_s7_cfg_data_db_no*) to make it easier to select the size of the data buffer. Using this number, the USER can select the appropriate "l2_mem_alloc_db***x*** ()" function from AMPRO2 memory management.
  When using this technique, the USER should not forget to transfer the length specification from the SLCB (entry: *max_s7_cfg_data_len*) together with the buffer.

5.2.4.13 write_diag_data_to_pda
*(SLCB_PTR, Unsigned8 L2_DATA_ATTR **)*

Diagnostic data must always be transferred with "long" consistency. For diagnoses, data communication is not triggered by the USER but by AMPRO-DPM during the transition from a diagnostic state to any other state *before* the "state_report ()" function is called. As soon as a new diagnosis must be reported, AMPRO-DPM calls the "write_diag_data_to_pda ()" CBF. The USER receives a pointer to the SLCB as a parameter of the CBF. The pointer to the process data area of the diagnoses of this slave (entry: *diag_data_ptr*) and the current length of the diagnoses (entry: *act_diag_data_len*) is entered there. The USER also receives a pointer-pointer to a buffer with the current diagnostic data. When Siemens DP slaves are involved, the diagnoses are reformatted to the standard DP format. See section on diagnosis conversion starting on page 149. The USER must provide for the transfer of the diagnoses from the buffer to the process data area. There are two ways to accomplish this.

- ☞ If the process data area is a shared memory area (i.e., dual-port RAM or similar), the USER activates consistency control for the area specified by the process data pointer. Using the pointer-pointer, the USER then copies all data from the buffer to the process data area. The SLCB contains the required length. The USER then deactivates consistency control and concludes the CBF.

  Caution:
  Activation and deactivation of the consistency lock during the "write_diag_data_to_pda ()" must never be allowed to increase the run time of the CBF. See also section on consistency assurance starting on page 70.

- ☞ If, for example, the data must be sent with an operating system message, the buffer can also be sent directly. In this case, the USER must obtain a new buffer from AMPRO2 memory management which the USER then enters in the buffer pointer. At the end of the CBF, AMPRO-DPM expects a buffer of any content to be valid. To make it easier to select the data buffer size, the buffer's number is entered in the SLCB (entry: *diag_data_db_no*). This number can be used by the USER to select the appropriate "I2_mem_alloc_db**x** ()" function of AMPRO2 memory management.
  When using this technique, the USER should not forget to transfer the length specification from the SLCB together with the buffer.

Since the "write_diag_data_to_pda ()" CBF is only called when new diagnostic data are available, the *new_diag_data* entry in the SLCB can only have the value TRUE at this time. This function does not require additional parameters (e.g., a word for status messages and similar).

5.2.4.14 clear_cons_input_data
(DPM_PROC_UNSIGNED8_PTR, unsigned8)

When "short" consistency is set for a memory area, the ASPC2 automatically ensures consistency, using HW control logic. When a slave fails, AMPRO-DPM must clear the inputs of this slave via software. Previously, this was done without consistency. To correct the problem, the user must provide the new AMPRO-DPM version with a CBF which can clear input data while maintaining consistency. This CBF is only called for slaves which are equipped with inputs with "short" consistency. As the parameter of the CBF, AMPRO-DPM transfers a pointer to the input data area to be cleared and its length. The following function can be used as an example. Depending on the design of the consistency control logic, the function may have to be time-optimized. This can be accomplished by using word-access clearing within the for-loop or by programming the entire function in Assembler.

```
void clear_cons_input_data (DPM_PROC_UNSIGNED8_PTR, Unsigned8);

...

void clear_cons_inputs_data (DPM_PROC_UNSIGNED8_PTR input_byte_ptr, Unsigned8 len)
{
        /* USER-related activation of the HW consistency control logic */
        activate_input_short_cons ();

        for (; len--;) *input_byte_ptr++ = (Unsigned8) 0;

        /* USER-related deactivation of the HW consistency control logic */
        deactivate_input_short_cons ();
}
```

5.2.4.15 consistency_update_done (Unsigned16)

AMPRO-DPM only calls the "consistency_update_done ()" CBF when the USER previously started the "consistency_update ()" function (see page 118 ff.) because of a slave with inputs with "long" consistency or with inputs in Buffered_Mode. See section on consistency assurance starting on page 70. This call informs the USER that the last "consistency_update ()" call has been executed and a complete update cycle has been performed for the inputs.

Possible values for the status word:

DPM_OK	Service executed correctly

5.2.4.16 asic_int_disable (void)

The USER must provide AMPRO-DPM with this function to disable the ASIC interrupts of the ASPC2. This ensures reliable operation of AMPRO-DPM since the L2 calls may not be interrupted by other L2 requests. This also applies to internal L2 requests from AMPRO-DPM. Multiple calls of this function must be logged to ensure that the enable function which is called just as frequently actually enables the ASIC interrupt again. See also section on disable times on page 79.

5.2.4.17 asic_int_enable (void)

The USER must provide AMPRO-DPM with this function to enable the ASIC interrupts of the ASPC2. This ensures reliable operation of AMPRO-DPM since the L2 calls may not be interrupted by other L2 requests. This also applies to internal L2 requests from AMPRO-DPM. When this function is called more than once, each call must evaluate the call log of the interrupt disable function to ensure that the enable function which is called just as frequently actually enables the ASIC interrupt again. See also section on disable times on page 79.

5.2.4.18 error (ERRCB)

5.2.4.18.1 Description

When an error which cannot be handled by AMPRO-DPM occurs at any time during slave processing by AMPRO-DPM or AMPRO2, AMPRO-DPM calls the "error ()" function. This corresponds to a system crash for AMPRO-DPM during which one final error message can be sent. The USER receives a block providing

more details on the type of error. Using the information in this message, the USER can address LEDs and so on. The USER should always save this block for service personnel.

Caution:

With this function, the block is transferred directly and not with a pointer to a block as is done with the other functions.

5.2.4.18.2 ERRCB

5.2.4.18.2.1 Header

Type	Designation	Direction of Access			
		USER		DPM	
		L	S	L	S
DPM_IFA_DATA_ATTR *	next_blk_ptr			X	X
DPM_IFA_DATA_ATTR *	prev_blk_ptr			X	X
Unsigned8	opcode (irrelevant)				
Unsigned8	subsystem (irrelevant)				
Unsigned16	id_ptr (irrelevant)				

The header of the ERRCB is only used to adapt the layout of this block to the other control blocks. Since no values are entered, the USER cannot evaluate anything there.

5.2.4.18.2.2 General Portion

Type	Designation	Direction of Access			
		USER		DPM	
		L	S	L	S
Unsigned16	Firmware package	X		X	X
Unsigned16	Function group of the firmware package	X		X	X
Unsigned16	Status of the function group	X		X	X
Unsigned16	Error number	X		X	X
Unsigned16	Error detail	X		X	X

Firmware package:

- ☞ AMPRO2
- ☞ AMPRO-DPM

Function groups of the firmware package:

- ☞ For AMPRO2:
  - ➢ IFA
  - ➢ MAC
- ☞ For AMPRO-DPM:
  - ➢ DPM-USIF
  - ➢ DPM-SLSM
  - ➢ DPM-DATR

Status of the function group:

Since this information applies to internal firmware program states, it is only of interest to the developer.

Error number:

This is an error identifier containing information on the type of error.

Error detail:

This is a supplement to the error identifier providing more details on the error.

For the identifiers for the firmware package and the function groups, see the section on coding rules starting on page 154. A series of error numbers and more details on the error numbers are defined for each of the packages and function groups. These are contained in the "COMMON\DP_ERROR.H" file and its files. All current identifiers can be read-accessed from these files. To keep preparation and maintenance expenses for these specifications low, we have not included a copy of these entries here.
Additional USER error identifiers can be entered under new headers which are linked as #include statements in the global "\COMM_DEV\DEV_ERR.H" header. All error identifiers for the other components of the IM 308-C firmware are defined in this header file, for example. These can be used as examples for your own entries.

5.2.5 Slave Families Supported by AMPRO-DPM

5.2.5.1 Standard DP Slaves

In its function as a class-1 master in accordance with the PROFIBUS-DP standard, AMPRO-DPM supports all slaves which comply with this standard.
Since this standard provides a very detailed description of the required behavior of a standard DP slave, it is not necessary to discuss this topic here. Storage of the diagnostic data and all other formatted data structures conform to the structures specified by the PROFIBUS-DP standard.

Standard DP slaves which are equipped with the LSPM PROFIBUS ASIC from Siemens require special treatment.
Although the LSPM supports the protocol stack of a DP standard slave, the PRM_UNLOCK service cannot be used. To be able to log off the slave from the master despite this, the PRM_UNLOCK procedure described for the Siemens DP slaves is used instead. This special treatment can be disregarded by the USER since the same PRM_UNLOCK status designator is used for this PRM_UNLOCK procedure.

5.2.5.2 Siemens DP Slaves

In addition to the standard DP slaves, AMRPO-DPM supports Siemens DP slaves in accordance with ET 200 communication specifications. There are two types of Siemens DP slaves.

- ☞ Slaves with the SPC PROFIBUS ASIC (e.g., the ET 200U, the IM 318-M and several other special slaves of various manufacturers). This type of slave will now be called the SPC slave.
- ☞ Slaves with the SPM PROFIBUS ASIC (e.g., the ET 200K or IM 418-B, the ET 200B and several other special slaves of various manufacturers). This type of slave will now be called the SPM slave.

Since Siemens DP slaves require a subset of the functionalities prescribed by the PROFIBUS-DP standard, Siemens DP functionality is the same as that of standard DP procedures. Since Siemens DP slaves always assume the same states as standard DP slaves, the "state_report ()" CBF (see page 129 ff.) is called the same as for the standard DP transitions in states. However, the telegram formats and contents used for the

individual states differ considerably from the standard DP telegrams. In addition, different types of telegrams are sometimes required for the individual states depending on the type of Siemens DP slave.

Most of these differences are not of concern to the USER since AMPRO-DPM transfers most of the Siemens DP specifications to the user interface in standard DP format. For this reason, only a few of the most important differences will be discussed here. For detailed information, see the ET 200 communication specifications.

5.2.5.3 DP S7 Slaves

The required configuration for certain types of S7 slaves must requested immediately after the startup phase. These configuration data must be supplied to the USER for evaluation. The procedures described below are used to implement this functionality.

In addition to the purely standard DP slaves, AMPRO-DPM supports DP S7 slaves which conform to the PROFIBUS-DP standard but does not generate a DPX protocol. These S7 slaves are handled the same as standard DP slaves. In addition to DP S7 slaves which conform to the PROFIBUS-DP standard, DP S7 slaves which can generate a logical configuration during startup are also supported. The primary characteristic of these slaves is that the current configuration of the slave is requested by the DP master during startup.

5.2.5.3.1 State Transitions of the Master

Standard DP slaves are cyclically informed of the current master state via the MA_CLEAR or MA_OPERATE Global_Control command.
Siemens DP slaves expect a CLEAR message only once when the master assumes the MA_CLEAR state. Afterwards, the duration of the MA_CLEAR master state and the transition from MA_CLEAR to MA_OPERATE are irrelevant for Siemens DP slaves.
For this reason, an SDN_HIGH telegram without user data is sent once to every configured Siemens DP slave when the master assumes MA_CLEAR. The slaves do not receive a new CLEAR telegram until the master changes from MA_CLEAR to MA_OPERATE again. While the master is in the MA_CLEAR state, the Siemens DP slaves also only receive output data cleared to 00_H regardless of the status of the process data of the slave. Current outputs are not sent again until after the transition of the master to MA_OPERATE.

5.2.5.3.2 Slave Handler with Parameterizing Master

Fundamental handling of the state machine for SPC slaves corresponds to the procedure described in the PROFIBUS-DP standard. See "state_report" () CBF starting on page 129. Shared_Input operation cannot be used with Siemens DP slaves. In addition, SPM slaves must be handled slightly differently. These differences are described below.

5.2.5.3.2.1 Slave State Transitions during Normal Operation for SPM Slaves

Since these slaves do not recognize the CFG state, a direct transition is made from PRM to DIAG2. In addition, these slaves do not use the DATA_NA state. The following figure illustrates this behavior.

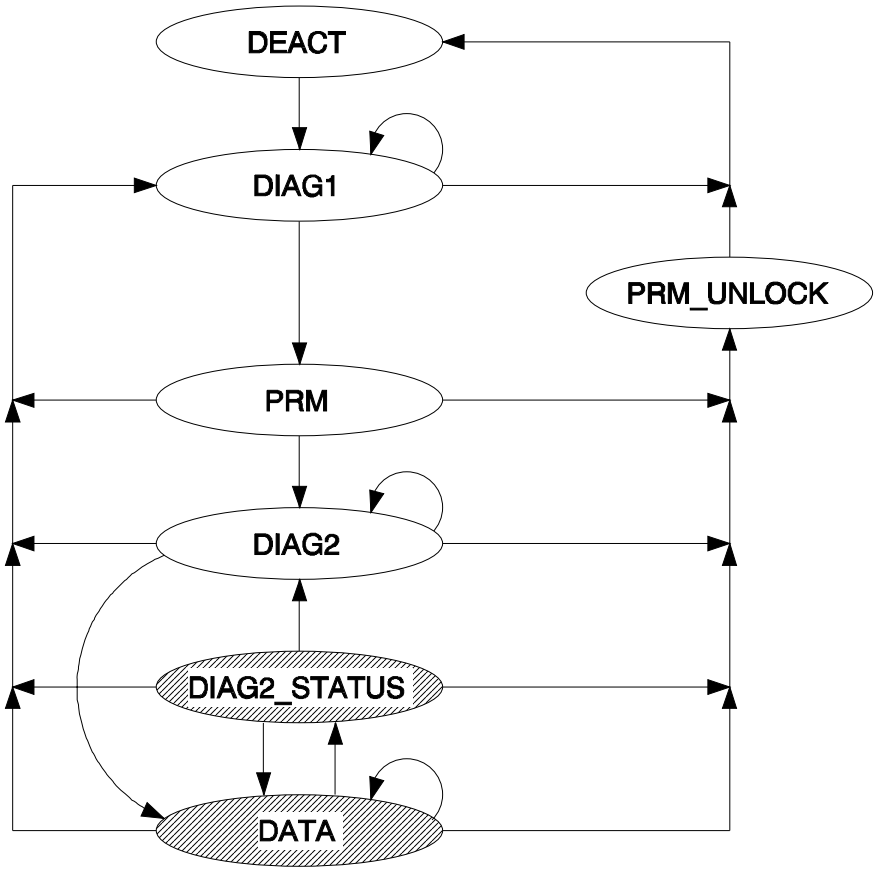


Figure 11: State transitions of SPM slaves with parameterizing master during normal operation

5.2.5.3.2.2 Slave State Transitions in AUTOSTOP Master Mode for SPM Slaves

During transitions to AUTOSTOP master mode, only the CFG slave state is omitted for SPM slaves. The following figure illustrates this behavior.

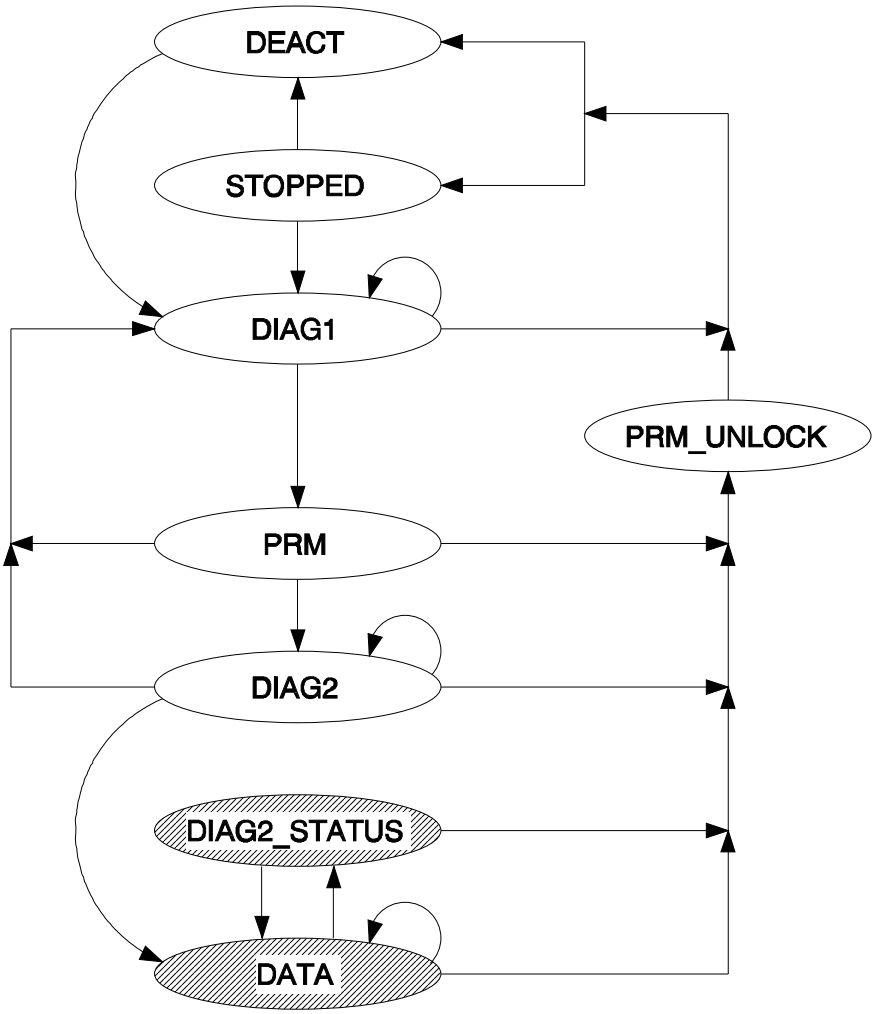


Figure 12: State transitions of SPM slaves with parameterizing master in AUTOSTOP master mode

5.2.5.3.3 PRM_UNLOCK

In the state machine, various procedures are used to achieve the PRM_UNLOCK state. For standard DP slaves which are not equipped with an LSPM2 ASIC, the PRM_UNLOCK service is handled as stated in the PROFIBUS-DP standard.

Siemens DP slaves and standard DP slaves with the LSPM2 ASIC achieve the PRM_UNLOCK state when the threshold monitoring time expires. This threshold monitoring time is set to the smallest possible value. A parameter telegram with the entries "LOCK" and "WD=1" is sent for this purpose to the appropriate slaves.

This procedure ensures reliable release of previous slave families to a standard DP master.

This special treatment can be disregarded by the USER since the same state designator (i.e., PRM_UNLOCK) is used for this PRM_UNLOCK procedure.

5.2.5.3.4 Diagnosis Conversion

All diagnostic data are provided to the USER in standard DP format. To obtain this uniform representation of the diagnoses, the diagnoses of the Siemens DP slaves must be converted to standard DP format. There are two different diagnosis formats for Siemens DP slaves (i.e., the SPC slave format and the SPM slave format).
The following section explains how the diagnoses are converted for the two types of slaves.

5.2.5.3.4.1 Specifications for Both Types of Slaves

Designation of the data is shown below.

	MSB							LSB
Byte 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
:	:	:	:	:	:	:	:	:

The first byte of a field has the number 0, the second byte has the number 1 and so on. A single bit is specified with the array method. Bit 5 of byte 3 is designated as "3.5", for example. The LSB of byte X is always designated as "X.0". The MSB of byte X is always designated as "X.7". The defined designations of the bits and bytes are sometimes used instead of the numbers, but the syntax is always retained.

When any type of Siemens DP slave does not respond or no longer responds, the same diagnosis is stored as for a standard DP slave: Length: 6 bytes; Stationstatus_1.Station_Non_Existent (0.0) set; bits 0.1 to 0.7 and bytes 1 to 5 cleared.

Siemens DP slaves which do not have diagnostic capability only generate the diagnoses for "station not operating correctly" or "station has failed".

5.2.5.3.4.2 Handling of SPC Slaves

Since SPC slaves generate a standard DP diagnosis with a length of at least 13 bytes, all bits which are not mentioned here always have a value of 0B. Conversion of the individual SPC diagnoses is shown below.

SPC Diagnosis	⇨	***Standard DP Diagnosis***
Station diagnosis. Parameterization error (0.2) = 1B	⇨	Diag.Cfg_Fault (0.2) = 1B Diag.Ext_Diag (0.3) = 1B Diag.Stat_Diag (1.1) = 1B
Station diagnosis. Single error (0.3) = 1B	⇨	Diag.Ext_Diag (0.3) = 1B Diag.Stat_Diag (1.1) = 1B Device-related diagnosis (7.3) = 1B
Station status. Disable Port Write (1.7) = 1B	⇨	Diag.Master_Lock (0.7) = 1B Diag.Stat_Diag (1.1) = 1B
Station diagnosis. Station cannot be controlled (0.1) = 1B ***and*** Station status. Disable Port Write (1.7) = 0 B	⇨	Diag.Station_Not_Ready (0.1) = 1B Diag.Stat_Diag (1.1) = 1B
Station status. AG100 Slow Mode (1.4) = 1B	⇨	Device-related diagnosis (7.0) = 1B
Station diagnosis. Load voltage missing (0.4) = 1B ***or*** Station diagnosis. Incorrect output activation (0.5) = 1B ***or*** Station diagnosis. Festo (0.6) = 1B	⇨	Diag.Ext_Diag (0.3) = 1B Device-related diagnosis (7.4) = 0.4 (from ←) Device-related diagnosis (7.5) = 0.5 (from ←) Device-related diagnosis (7.6) = 0.6 (from ←)
Station status. Watchdog on (1.3) = 1B	⇨	Diag.WD_On (1.3) = 1B
Station status. Station type (1.2-1.0) ≠ 001B (Station type IM 318-B)	⇨	Diag.Prm_Fault (0.6) = 1B Diag.Stat_Diag (1.1) = 1B

The following specifications also apply.

- ☞ Bit 1.2 of the standard DP diagnosis (station is a DP slave) is permanently set to 1B.
- ☞ Diag.Ext_Diag_Overflow (2.7) is permanently set to 0B for "normal" SPC slaves. If the slave supplies manufacturer-related diagnoses, this bit is also set when more than the expected number of diagnoses are sent by the slave.
- ☞ If stationsstatus.Disable Port Write (1.7) = Diag.Master_Lock (0.7) = 1B, the address 00H is entered in Diag.Master_Add (byte 3). Otherwise the current station address of the master is entered.
- ☞ The Ident_Number from the user-related portion of the slave parameter record (Slave_User_Data) is entered in Motorola format in Ident_Number (bytes 5 and 6; byte 5 = Ident_Number_High and byte 6 = Ident_Number_Low).
- ☞ 02H is permanently entered as header for the device-related diagnosis (byte 6) since the device-related diagnosis has a fixed user length of one byte.
- ☞ 45H is permanently entered as header for the identifier-related diagnosis (byte 8) since the identifier-related diagnosis has a fixed user length of 4 bytes.
- ☞ The module diagnoses (bytes 2 to 5) are transferred as the identifier-related diagnosis to bytes 9 to 12 of the standard DP diagnosis. When converted to standard DP format, this results in a total length of 13D bytes for the diagnosis for "normal" SPC slaves.
- ☞ Some SPC slaves supply more than 6D bytes of diagnosis. These up to 26D additional bytes of data are added to the previously written data with a second header for "device-related diagnoses." When converted to standard DP format, this results in a total length of 15D to 42D bytes of diagnoses when the master receives 7D to 32D bytes of diagnosis from the SPC slave.
- ☞ All bits which are not explicitly set (i.e., 1B) are cleared (i.e., 0B).

5.2.5.3.4.3 Handling of SPM Slaves

When the slave is an SPM station, a fixed value (07H) is entered in byte 6 as the header byte for device-related diagnosis whose user length is 6D bytes. The identifier-related diagnosis is omitted. The first byte of the device-related diagnosis (byte 7) is a group diagnosis. Byte 8 is always 00H. Bytes 9 to 12 indicate a single diagnosis. When modules without diagnostic capability are involved, bytes 7 to 12 are always cleared (i.e., 00H). The others can only handle the group diagnosis or the single diagnosis. Unused areas are always cleared (i.e., 00H). When converted, the total length of the diagnosis is the same as that of "normal" SPC slaves (i.e., fixed at 13D bytes).

The two status bytes (i.e., the last two bytes of the received Data_unit telegram) from the received diagnostic telegram of the SPM slave are determined for the entries in bytes 7 to 12.

...	...	***Status0***	***Status1***

The actual value of the high byte of the SPM hardware identification word (i.e., SPM_HWIdent_High) is generated from the above. In this actual SPM_HWIdent_High, the bits are assigned in accordance with the bits of status0 and status1 as shown below.

Status0 and Status1	⇨	***Actual SPM_HWIdent_High***
Status0.Code0 (0.6) = 1B Status0.Code1 (0.7) = 1B	⇨	Code0 (0.0) = 1B Code1 (0.1) = 1B
Status1.Type0 (1.4) = 1B Status1.Type1 (1.5) = 1B Status1.Type2 (1.6) = 1B	⇨	Type0 (0.2) = 1B Type1 (0.3) = 1B Type2 (0.4) = 1B
Status1.TS_Prom (1.7) = 1B	⇨	TS_Prom (0.5) = 1B
Status1.Auto_Baud (1.1) = 1B	⇨	Auto_Baud (0.6) = 1B
-		0.7 = 0B

The SPM subtype is then generated from the "Slave_Type" entered in "general slave data" (Slave_Para_Data) of the slave parameter record.

Slave Type	⇨	***SPM Subtype***
SPM general, code 0	⇨	SPM subtype 0
SPM general, code 1	⇨	SPM subtype 1
SPM general, code 2 ET 200B ET 200C	⇨	SPM subtype 2
SPM general, code 3 ET 200K	⇨	SPM subtype 3

Further processing is performed in two steps. The first step depends on the subtype of the SPM. In the first step, an evaluation is performed of the actual SPM_HWIdent_High obtained from the data received and, when applicable, the diagnostic bits. The received status bytes are then evaluated.

Subtype 0:

☞ The SPM ident byte is determined from the telegram.

...	***Ident***	Status0	Status1

This SPM ident byte must correspond to the "SPM_HWIdent_Low" byte entered in the "user data" (i.e., Slave_User_Data) of the slave parameter record. In addition, the determined actual SPM_HWIdent_High must correspond to the "SPM_HWIdent_High" byte entered in the "user data" (i.e., Slave_User_Data) of the slave parameter record.

The following entries are made in the converted diagnosis when at least one of these two conditions is not fulfilled.

- Diag.Cfg_Fault (0.2) = 1_B
- Diag.Stat_Diag (1.1) = 1_B

☞ Since subtype 0 SPM slaves have no further diagnostic capabilities, the remaining bytes 7 to 12 are always cleared (00_H).

Subtype 1:

☞ Since the telegram of a subtype 1 SPM slave does not contain an ident byte, only the determined actual SPM_HWIdent_High is checked for plausibility with the specified value.

...	Status0	Status1

When a discrepancy occurs, the same procedure is followed as for subtype 0.

☞ Since subtype 1 SPM slaves have no further diagnostic capabilities, the remaining bytes 7 to 12 are always cleared (00_H).

Subtype 2:

☞ Although the telegram of a subtype 2 SPM slave does contain an ident byte, this byte is the seventh byte as seen from the end of the telegram.

...	***Ident***	Port F	Port G	Port H	Port I	Status0	Status1

Further evaluation of the ident byte and the status bytes is the same as for subtype 0.

☞ In addition to the ident byte, subtype 2 SPM slaves provide diagnoses of port bytes F to I.

...	Ident	***Port F***	***Port G***	***Port H***	***Port I***	Status0	Status1

To evaluate these ports, an additional branch is required based on the "Slave_Type" entered in the "general slave data" (i.e., Slave_Para_Data) of the slave parameter record.

- *ET 200B with diagnosis*
  With this SPM station, bits 0 to 7 of the four port bytes are negated in bits and entered in byte 7 (i.e., first byte of the device-related diagnosis: group diagnosis).

Port Byte	⇨	***Byte 7 of the Standard DP Diagnosis***
Port F / bit 0 = 0B	⇨	7.0 = 1B
Port F / bit 7 = 0B	⇨	7.1 = 1B
Port G / bit 0 = 0B	⇨	7.2 = 1B
Port G / bit 7 = 0B	⇨	7.3 = 1B
Port H / bit 0 = 0B	⇨	7.4 = 1B
Port H / bit 7 = 0B	⇨	7.5 = 1B
Port I / bit 0 = 0B	⇨	7.6 = 1B
Port I / bit 7 = 0B	⇨	7.7 = 1B

If, after this conversion, byte 7 of the standard DP diagnosis is not equal to 00H, the Diag.Ext_Diag (0.3) bit is also set.
Since this ET 200B does not have single-diagnosis capability, bytes 8 to 12 are always cleared (00H).

- *ET 200B without diagnosis*
  Since this SPM station has neither group nor single-diagnosis capability, bytes 7 to 12 of the standard DP diagnosis are always cleared (00H).

- *All other types of SPMs*
  For all other SPM slaves, the four port bytes are negated and entered in bytes 9 to 12. These bytes correspond to the single diagnosis.

Port Byte	⇨	***Byte of the Standard DP Diagnosis***
Port F / negated	⇨	Byte 9
Port G / negated	⇨	Byte 10
Port H / negated	⇨	Byte 11
Port I / negated	⇨	Byte 12

When at least one of bytes 9 to 12 is not equal to 00H, the Diag.Ext_Diag (0.3) bit is also set. Since these slaves do not supply group diagnoses, bytes 7 and 8 of the standard DP diagnosis are always cleared (00H).

This procedure can also be used for the ET 200C. Although it does not supply a single diagnosis, FFH is always entered in all port bytes which produces the message "no diagnosis."

Subtype 3:

☞ Since the telegram of a subtype 3 SPM slave does not contain an ident byte, only the determined actual SPM_HWIdent_High is checked for plausibility with the specified value.

...	Port F	Port G	Port H	Port I	Status0	Status1

Further evaluation of the status bytes is the same as for subtype 0.

- ☞ Evaluation of the port bytes is the same as the evaluation for subtype 2 slaves with the identifier "other SPM slave types."

Now that the slave subtype has been processed, the following bits of the status bytes must be converted in the second step.

Status Byte	⇨	***Standard DP Diagnosis***
Status0.Disable_Port_Write (0.5) = 1_B	⇨	Diag.Master_Lock (0.7) = 1_B Diag.Stat_Diag (1.1) = 1_B
Status1.Watch_Dog_on (1.1) = 1_B	⇨	Bit 1.3 (Diag.WD_On) = 1_B

In conclusion, the following settings must also be made.

- ☞ Bit 1.2 of the standard DP diagnosis (i.e., the station is a DP slave) is permanently set to 1_B.
- ☞ Diag.Ext_Diag_Overflow (2.7) is permanently set to 0_B.
- ☞ If Status0.Disable Port Write (0.5) = Diag.Master_Lock (0.7;) = 1_B, the address 00_H is entered in Diag.Master_Add (byte 3). If not, the current station address of the master is stored there.
- ☞ The Ident_Number from the user-related portion of the slave parameter record (i.e., Slave_User_Data) is entered in Motorola format (i.e., byte 5 = Ident_Number_High and byte 6 = Ident_Number_Low) in the Ident_Number (bytes 5 and 6).
- ☞ All other bits which are not explicitly set (i.e., 1_B) are cleared (i.e., 0_B).

5.2.6 Coding Rules

This section provides the numerical values of the individual definitions and other information relevant to implementation. For further information, see the applicable header files in the directories "\COMMON" and "\COMM_DEV". All entries in this section have be copied to the specifications.

5.2.6.1 Global Definitions

5.2.6.1.1 Definition of the ZERO Pointer

In AMPRO-DPM, ZERO is defined as the void pointer with a value of 0_H. If necessary, the USER can change this definition is his programs. Use of the following procedure is recommended.

AMPRO-DPM:

```
#define    DP_NULL              (void *) 0          /* File: DP_DEFMA.H */
```

5.2.6.1.2 Definitions for Boolean Values

In the DP standard, the following two values are defined for boolean variables. See also section on variable types starting on page 63. Values other than these are not used by AMPRO-DPM.

```
#define    DP_TRUE    (Unsigned8)     0xff          /* File: DP_TYPES.H */
#define    DP_FALSE   (Unsigned8)     0x00
```

5.2.6.2 Slave-Related Identifiers

5.2.6.2.1 Slave States

These definitions are located in the "\COMMON\DPM_COMM.H" file.

Identifier	*Code*
SL_DEACT	(Unsigned8) 01H
SL_STOPPED	(Unsigned8) 0DH
SL_DIAG1	(Unsigned8) 02H
SL_PRM	(Unsigned8) 03H
SL_CFG	(Unsigned8) 04H
SL_DIAG2	(Unsigned8) 05H
SL_DIAG2_STATUS	(Unsigned8) 0CH
SL_DATA	(Unsigned8) 06H
SL_DATA_NA	(Unsigned8) 07H
SL_PRM_UNLOCK	(Unsigned8) 08H
SL_SIO_GET_CFG	(Unsigned8) 09H
SL_SIO_DIAG2	(Unsigned8) 0AH
SL_SIO_RD_IO	(Unsigned8) 0BH
SL_STS_PRM	(Unsigned8) 10H
SL_S7_GET_CFG	(Unsigned8) 11H

5.2.6.2.2 Slave Types

These definitions are located in the "\COMMON\DPM_COMM.H" file. The MSB of the identifier is used to distinguish between standard DP slaves (MSB = 0B) and Siemens DP slaves (MSB = 1B).

Type Series	*Identifier*	*Code*
Standard DP slaves	DPM_SL_TYPE_DP	(Unsigned8) 00H
	DPM_SL_TYPE_LSPM2	(Unsigned8) 10H
	DPM_SL_TYPE_DP_S7	(Unsigned8) 11H
Siemens DP slaves	DPM_SL_TYPE_ET200U	(Unsigned8) 80H
	DPM_SL_TYPE_ET200B_DIAG	(Unsigned8) 81H
	DPM_SL_TYPE_ET200B_NO_DIAG	(Unsigned8) 82H
	DPM_SL_TYPE_SPM_0	(Unsigned8) 83H
	DPM_SL_TYPE_SPM_1	(Unsigned8) 84H
	DPM_SL_TYPE_SPM_2 (incl. ET 200C)	(Unsigned8) 85H
	DPM_SL_TYPE_SPM_3 (incl. ET 200K)	(Unsigned8) 86H

5.2.6.2.3 Slave Operating Modes

These definitions are located in the "\COMMON\DPM_COMM.H" file. The operating modes of the slave are set with the "set_slave_mode ()" command. The *Global_Control_Command* parameter can be used to activate or deactivate the SYNC and FREEZE functions. When several functions are to be executed simultaneously, the identifiers must be or-linked. The function is deactivated when the bits for activation and deactivation of a function are set at the same time.

Identifier	*Code*
SL_SYNC	(Unsigned8) 20H
SL_UNSYNC	(Unsigned8) 10H
SL_FREEZE	(Unsigned8) 08H
SL_UNFREEZE	(Unsigned8) 04H

5.2.6.3 Master-Related Identifiers

5.2.6.3.1 AMPRO-DPM Functions

These definitions are located in the "\COMMON\DPM_COMM.H" file. The identifiers are entered in the *opcode* field of the individual blocks.

Identifier	*Code*
DPM_INIT	(Unsigned8) 01H
DPM_ADD_SLAVE	(Unsigned8) 02H
DPM_WITHDRAW_SLAVE	(Unsigned8) 03H
DPM_RESTART_SLAVE	(Unsigned8) 0BH
DPM_SET_MASTER_MODE	(Unsigned8) 04H
DPM_SET_SLAVE_MODE	(Unsigned8) 05H
DPM_TIMER_EXPIRED	(Unsigned8) 06H
DPM_INPUT_UPDATE	(Unsigned8) 07H
DPM_OUTPUT_UPDATE	(Unsigned8) 08H
DPM_MARK_CYCLE	(Unsigned8) 09H
DPM_SET_SLAVE_ADDRESS	(Unsigned8) 0AH
DPM_ERROR	(Unsigned8) FFH

5.2.6.3.2 Master States

These definitions are located in the "\COMMON\DPM_COMM.H" file. Only the MA_STOP, MA_CLEAR and MA_OPERATE states are defined for AMPRO-DPM. The USER also recognizes the OFFLINE state for AMPRO-DPM.

Identifier	*Code*
MA_OFFLINE	(Unsigned8) 10H
MA_STOP	(Unsigned8) 40H
MA_CLEAR	(Unsigned8) 80H
MA_OPERATE	(Unsigned8) C0H

5.2.6.4 Return Values/Error Identifiers

5.2.6.4.1 Identifiers for the Firmware Packages

These definitions are located in the "\COMMON\DP_ERROR.H" file.

Identifier	*Code*
ERR_AMPRO2	(Unsigned16) 0001H
ERR_DPM	(Unsigned16) 0010H

5.2.6.4.2 Identifiers for the Function Groups of the Firmware Packages

These definitions are located in the "\COMMON\DP_ERROR.H" file.

Identifier	*Code*
ERR_AMPRO2_IFA	(Unsigned16) 0001H
ERR_AMPRO2_MAC	(Unsigned16) 0002H
ERR_DPM_USIF	(Unsigned16) 0001H
ERR_DPM_SLSM	(Unsigned16) 0002H
ERR_DPM_DATR	(Unsigned16) 0003H

6 Parameter Module Description

The description of the parameter module includes the bus, master and slave parameters for the application.

Current version of the parameter data:	V 1.01

6.1 Data Layout

6.1.1 General

Storage structure and the layout of the parameter module data are based on the PROFIBUS-DP /7/ standard publication.

The data are stored in blocks. The blocks are located in succession in the memory area without gaps. They always begin at word boundaries.

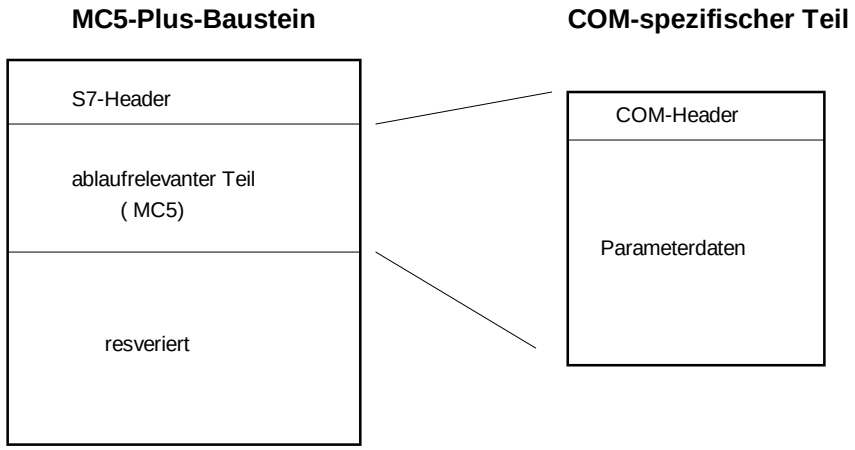


Figure 13 : Layout of an MC5-Plus block

⇨ COM uses or writes the following portions of an MC5-Plus block.

- S7 header
- Processing-related portion

⇨ The portion of the block concerning processing consists of 2 parts.

1. COM-related header
2. COM data

⇨ COM data are either parameters or operating system data.

6.1.2 Parameter Data

Although the layout of the parameter records in the data area follows the suggestion in the standard, certain conditions must be adhered to because of the ASIC. Data which are processed by word and sent directly from the module to the bus must start at even-numbered word addresses.

A DP master parameter record consists of two data records each with a different layout (i.e., bus parameter records and slave parameter records). (/7/)
The first block contains a pointer field for addressing each of the individual data records on the module.
When processing the data with import and export functions, it is assumed that the data are correct and complete (i.e., syntax and semantics are not checked).

Block 1: Pointer field

The pointer field is used to address the individual parameter records.
On the memory module, the pointer field contains an additional free area so that additional parameter records can be addressed. This is not reserved for storage in the binary file.

Block 2: Bus parameter record
The bus parameter record is always located in the 2nd block in the data area.
It consists of the following data structures.
- Bus parameters
- Master parameters (incl. host parameters)

Block 3-N: Slave parameter record 1-N
A slave parameter record consisting of the following data structures.
- General slave data
- Parameterization data
- Configuration data
- Address table
- User data

The following figure shows the layout of the parameter data.

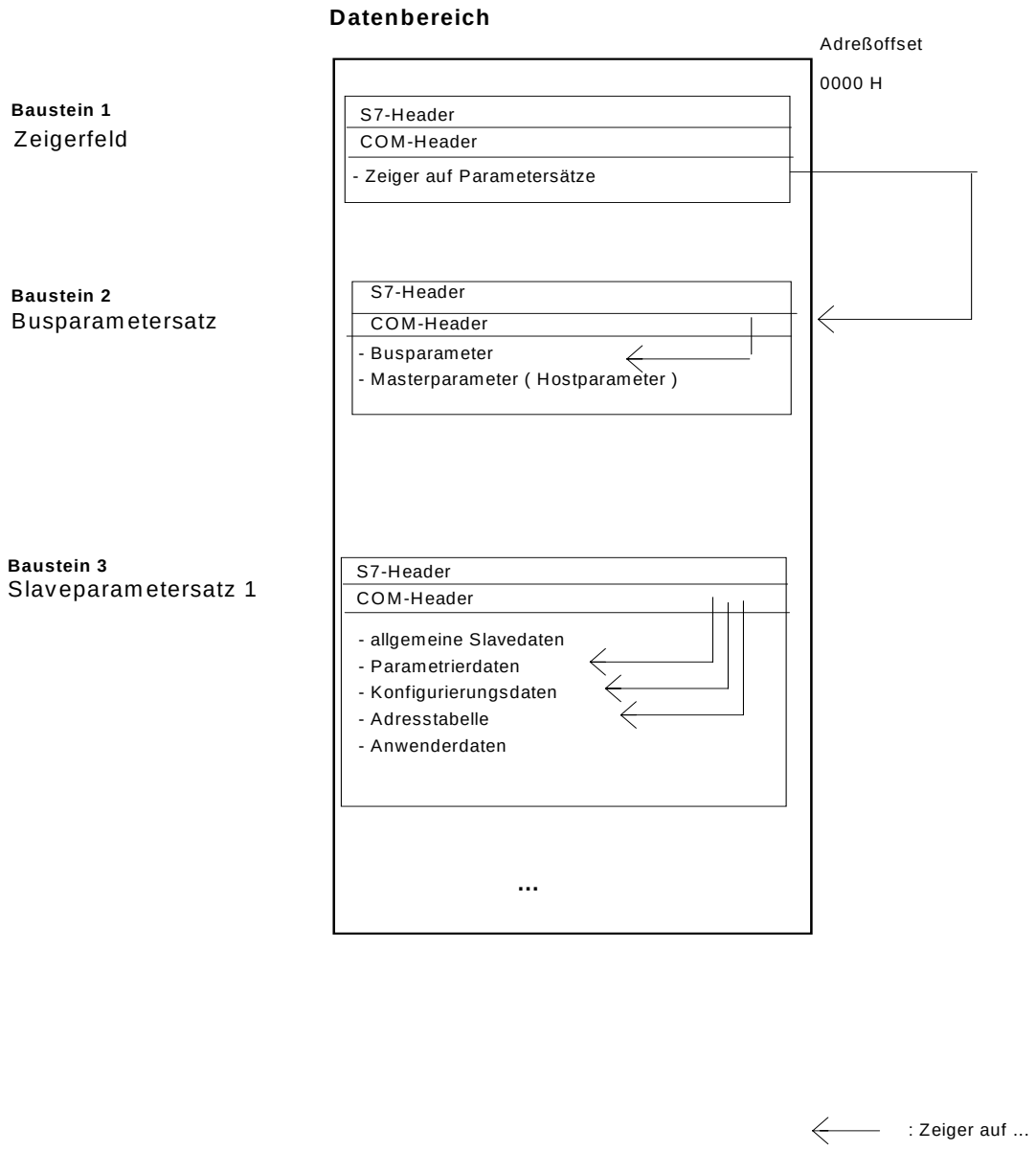


Figure 14 : Layout of the parameter data in the data area of the memory module

6.2 Data Storage

6.2.1 General

The following sequence of data words and Dwords applies to data storage in general.

In MOTOROLA format:
- Data words: High byte/low byte
- Storage of Dwords: High word/low word

In INTEL format:
- Data words: Low byte/high byte
- Storage of Dwords: Low word/high word

The PROFIBUS-DP /7/ standard applies to data transmission on the bus (i.e., the COM data must be transferred in Motorola format).

Due to the functions and hardware, the Intel format is used for internal COM data storage.

The following specifications apply to the layout of the data structures in general.

⇨ All length specifications in [bytes]

⇨ TRUE = 0x01

⇨ FALSE = 0x00

⇨ DP_TRUE = 0xFF

⇨ DP_FALSE = 0x00

⇨ Significance of the bit positions
0 : FALSE /No/Does not apply
1 : TRUE / Yes /Applies

6.2.2 Parameter Data

⇨ Configuration with COM ET 200 V5.0 is performed for the entire bus.
⇨ A program file can thus contain several master systems.
⇨ The parameter data of a master system are generated from a program file.

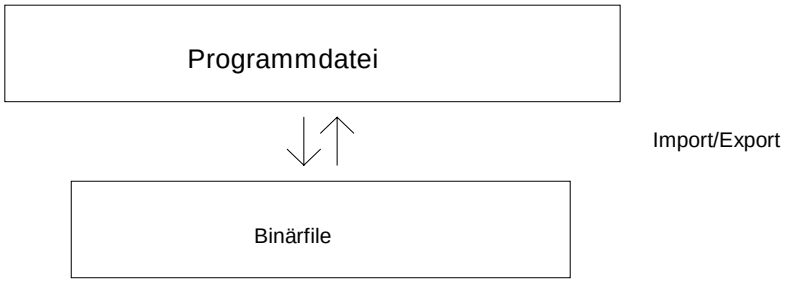


Figure 15: *Handling of parameter data*

6.2.2.1 Memory Module

Data storage:
The parameters for Prm_Data and Cfg_Data are stored in Motorola format. All other parameters are stored in Intel format.

6.3 Header

6.3.1 S7-Related Header

COM ET 200 V5.0 only uses the parameters shown.

Block type: S5_BSTTYP_DB (data block) [0x0A]

Block number:

COM uses the block/block numbers starting at 8000 H / 32768 dec.

Block Number	Parameter Data
8000 H + 0	Pointer field
8000 H + 1	Bus parameter data
8000 H + 2...	Slave parameter data 1...

Length of the total block

= S7 block header + COM-related data (user data) + HW identification data

... + reserved area

Length of the COM-related data = COM-related header + COM data

COM data = [pointer field | bus parameter record | slave parameter record]

Length of the COM identifier data = Fixed

6.3.1.1 S7 CPU Header

Address Offset BYTE	Designation	Contents	Value Range
0 to 1		Block identifier 0x7070	Unsigned16
2		Not used	Unsigned8
3		Not used	Unsigned8
4 to 5	bsttyp	Block type = S5_BSTTYP-DB	Unsigned16
6 to 7	bstnr	Block number = 0 to 65535	Unsigned16
8 to 11	geslen	Length of the total block incl. S7 header (low word-high word)	Unsigned32
12 to 31		Not used	Unsigned8
32 to 33	locdatalen	Length of reserved area	Unsigned16
34 to 35	mc5len	Length of COM-related data	Unsigned16

6.3.2 COM-Related Header

The *parameter pointer* and the station address entry depend on the data record identifier. Irrelevant pointers are assigned 0xFFFF.
The offset indicated starts at the COM header (i.e., offset 0).

Address Offset BYTE	Designation	Contents	Value Range
0	Data record identifier	3 = Pointer field 1 = Bus parameter record 2 = Slave parameter record	Unsigned8
1	Station address	Station addresses 1 to 125 - Master address - Master address - Slave address	Unsigned8
2 to *3*	*- Ptr_Field_Len* *- Bus_Para_Len* *- Slave_Para_Len*	*Pointer to the field with the length of the entire parameter record (length without header)*	*Unsigned16*
4 to 5	*- Ptr_Field* *- Bus_Para_Data* *- Slave_Para_Data*	*- Pointer to pointer field* *- Pointer to Bus_Para_Data* *- Pointer to Slave_Para_Data*	*Unsigned16*
6 to 7	- *- Master_User_Data* *- Slave_Prm_Data*	*- Disregard* *- Pointer to master parameter* *- Pointer to parameterization data*	*Unsigned16*
8 to 9	- *- Reserved* *- Cfg_Data*	*- Disregard* *- Reserved* *- Pointer to configuration data*	*Unsigned16*
10 to 11	- - *- Add_Tab*	*- Disregard* *- Disregard* *- Pointer to address table*	*Unsigned16*
12 to 13	- - *- Slave_User_Data*	*- Disregard* *- Disregard* *- Pointer to user data*	*Unsigned 16*
14 to 15	- - *- Reserved*	*- Disregard* *- Disregard* *- Reserved*	*Unsigned16*
16 to 17	- - *- Reserved*	*- Disregard* *- Disregard* *- Reserved*	*Unsigned16*
18 to 19	- *- Reserved* *- Reserved*	*- Disregard* *- Reserved* *- Reserved*	*Unsigned16*
20 to 21	- *- Reserved* -	*- Disregard* *- Reserved* *- Disregard*	*Unsigned16*
22 to 31		*Not assigned*	*Unsigned8*

6.4 Pointer Field

The pointer field is located in the first block of the data area.
A Dword pointer is used in the pointer field to address the other data records in the data area.
The pointers are stored in the pointer field in ascending order in the order in which they were configured.

Definition of the Dword pointer:

- Contains absolute address
- Offset = 0 (start of the memory area)
- Points to the COM-related header of the respective block

Contents of the pointer:

= FFFF FFFF H	Pointer is not assigned.
= 0000 0000 H	Pointer was cleared.
!= 0000 0000 H or == FFFF FFFF H	A data record is located under this address.

6.4.1 Pointer Field for Parameter Data

Address Offset BYTE	Designation	Contents	Value Range
0 to 1	*Ptr_Field_Len in bytes*	*Length of the entire pointer field including Ptr_Field_Len*	*Unsigned16*
2 to 3	Number of pointer entries	1 to 125	Unsigned16
4 to 7	*Pointer to data record 1 = bus parameter record*	*0000 1000 H *)*	*Unsigned32*
8 to 11	*Pointer to data record 2 = slave parameter record*	*0000 xxxx H*	*Unsigned32*
12 to	*...*		*Unsigned32*
	Pointer to data record N = slave parameter record	*0000 xxxx H*	*Unsigned32*

*) Example of the data record pointer in the memory module

ATTENTION: Data record pointer to COM-related header

6.5 Structures of the Parameter Data

6.5.1 Bus Parameter Record

The length of the entire parameter record specified in Bus_Para_Data includes the filler bytes required when parameter records must start at even-numbered addresses.

The lengths of the individual parameter data (i.e., Master_User_Data and so on) do not include filler bytes.

6.5.1.1 Bus Parameters (Bus_Para_Data)

Address Offset BYTE	**Designation**	**Contents**	**Value Range**
0 to 1	*Bus_Para_Len*	*Length of the entire bus parameter record including Bus_Para_Len*	*Unsigned16*
2	FDL_Add	Station address of the DP master	Unsigned8
3	Baud rate Baud rates 9, 8 and 7 are not in accordance with the DP standard.	9 = 12000 kbaud 8 = 6000 kbaud 7 = 3000 kbaud 6 = 1500 kbaud 4 = 500 kbaud 3 =187.5 kbaud 2 = 93.75 kbaud 1 = 19.2 kbaud 0 = 9.6 kbaud	Unsigned8
4 to 5	T_{SL}	Slot time	Unsigned16
6 to 7	minT_{SDR} = T_{RDY}	Station delay time	Unsigned16
8 to 9	maxT_{SDR}	Station delay time	Unsigned16
10	T_{QUI}	Quiet time	Unsigned8
11	T_{SET}	Setup time	Unsigned8
12 to 15	Ttr		Unsigned32
16	G	GAP update factor	Unsigned8
17	HSA	Highest station address	Unsigned8
18	retry_ctr	Retry counter	Unsigned8
19	Bp_flag	0: No change in operating mode when error occurs 128: Change in operating mode when error occurs	Unsigned8
20 to 21	Min_Slave_Intervall	Time interval between 2 accesses to the slave Time base: 100 [µsec]	Unsigned16
22 to 23	Poll_Timeout	Master-master communication monitoring time Time base: 1 [msec]	Unsigned16
24 to 25	Data_Control_Time	Max. data cycle time Time base: 10 [msec] = WdTimeout [msec] * 6 / time base	Unsigned16
26 to 31		Reserved	Unsigned8

6.5.1.2 Master Parameter (Master_User_Data)

The manufacturer-related data applicable to the bus parameter record are located here.

Address Offset BYTE	Designation	Contents	Value Range
0 to 1	*Master_User_Data_Len*	*Length of Master_User_Data including Master_User_Data_Len*	*Unsigned16*
2...	Master_Class2_Name	COM ET200 V05.00	Visible-String (33)
35...	Master_Class1_Name	IM 308-C V05.00	Visible-String (33)
68...	TRDY min. response delay	Slave pause	Unsigned16
70...	Tid1 idle time 1	Master pause	Unsigned16
72...	Tid2 idle time 2		Unsigned16
74...	DeltaTtr	Delta target rotation time	Unsigned32
78	SAP Master-slave communication	(SSAP)	Unsigned8
79	SAP Master-master communication	(SSAP and DSAP)	Unsigned8
80		Reserved	Unsigned8
81		Reserved	Unsigned8
82 to		Reserved	Unsigned16
84		Reserved	Unsigned8
85		Reserved	Unsigned8
86		Reserved	Unsigned8
87		Reserved	Unsigned8
88		Reserved	Unsigned8
89		Reserved	Unsigned8
90		Reserved	Unsigned8
91		Reserved	Unsigned8
92		Reserved	Unsigned8
93		Reserved	Unsigned8
94 to		Reserved	Unsigned16
96	Group_Sync_Byte		Unsigned8
97	Group_Freeze_Byte		Unsigned8
98	IM 308-C mode	Reserved	Unsigned8

99	Ident_Number_High	In acc. w. PNO / from ini. file	Unsigned8
100	Ident_Number_Low		Unsigned8
101		Reserved	Unsigned8
102...	WdTimeout	Trigger mon. time [t_bit] - system or - user-defined	Unsigned32
106...	IM 308-C firmware	Reserved (16 bytes)	Unsigned8
122...	DB_Len1	Buffer length Default: 32	Unsigned16
124...	DB_Number1	No. of buffers	Unsigned16
126...	DB_Len2	Buffer length Default: 128	Unsigned16
128...	DB_Number2	No. of buffers	Unsigned16
130...	DB_Len3	Buffer length Default: 186	Unsigned16
132...	DB_Number3	No. of buffers	Unsigned16
134...	DB_Len4	Buffer length Default: 244	Unsigned16
136...	DB_Number4	No. of buffers	Unsigned16
138...	Repeater	TRUE Repeater on bus FALSE No repeater on bus	Unsigned16
140		Reserved	Unsigned8
141		Reserved	Unsigned8
142...		Reserved	Unsigned16
144...	L4_Header_Len1	L4 header length Default: 0	Unsigned16
146...	APB_Number1	No. of APBs	Unsigned16
148...	L4_Header_Len2	L4 header length Default: 2	Unsigned16
150...	APB_Number2	No. of APBs	Unsigned16
152...	L4_Header_Len3	L4 header length Default: 4	Unsigned16
154...	APB_Number3	No. of APBs	Unsigned16
156...	L4_Header_Len4	L4 header length Default: 6	Unsigned16
158...	APB_Number4	No. of APBs	Unsigned16
160...	DPxMaxAuftrag	Max. no. of DPx jobs per DP cycle Default: 1	Unsigned16
161	Special functions	Bit 0: 0 = No AUTOSTOP 1 = AUTOSTOP Default: 0 Bit1-7: Reserved	Unsigned8
162		Reserved	Unsigned8
163		Reserved	Unsigned8
164		Reserved	Unsigned8

[Byteoffset 2...] MasterClass2Name:
[Byteoffset 35...] MasterClass1Name:

The versions are entered in accordance with the entries "COM_Version" and "HW_Version."

[Byteoffset 96] Group_Sync_Byte:
[Byteoffset 97] Group_Freeze_Byte:

Bit positions 0 to 7 correspond to groups 1 to 8.
The SYNC and FREEZE group options are entered by group (i.e., when a slave belongs to a group). SYNC and FREEZE are set to 0 when no group is defined.

EXAMPLE:
Slave 2 belongs to groups 6, 5, 2 and 1. The groups have the following options.

Group option for group:	**6**	**5**	**2**	**1**
Sync_Byte:	-	x	-	x
Freeze_Byte:	x	-	x	x

SYNC and FREEZE bytes:

Bit position:	7	6	5	4	3	2	1	0
Group:	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Sync_Byte: 0x11	0	0	0	1	0	0	0	1
Freeze_Byte: 0x23	0	0	1	0	0	0	1	1

[Byteoffset 102...] WDTimeout:
The factors WD_Faktor1 and WD_Faktor2 in Slave_Prm_Data are calculated from this trigger monitoring time.

6.5.1.2.1 DB Buffer Sizes and Number of Buffers

General:

The parameters for buffer size and number of buffers are required by AMPRO2 memory management.

The buffer sizes are divided into 4 categories.
The number of buffers per buffer size is preset with a default value of 0.

Size	**in [bytes]**	**Number**
Buffer 1	32	0 to N
Buffer 2	128	0 to N
Buffer 3	186	0 to N
Buffer 4	244	0 to N

When the export memory module is used, all slaves in the master system are examined and, when the conditions are fulfilled, the number of buffers is incremented by 1 for the appropriate buffer category. The size is rounded to the next higher number.

Calculation of buffer size and number of buffers:

Calculation of the number of buffers per buffer size is based on the following conditions.

Entries per slave:

- Consistency buffer:
  3 buffers for "long" consistency for inputs of the slave or for Buffered_Mode when a slave with only byte modules has exceeded the maximum telegram length for inputs
  The length of the buffer is determined by the length of the input bytes used.
  3 buffers for "long" consistency for outputs of the slave
  The length of the buffer is determined by the length of the output bytes used.

- Diagnostic buffer:
  Always 1 buffer with the length DiagRespLen (see Slave_User_Data)
  1 buffer with the length DiagDataLen (see Slave_User_Data) for non standard DP slaves

- Parameterization data buffer:
  1 buffer with the length of the parameterization data for SPM slaves

- Configuration data buffer:
  1 buffer with the length of the configuration data for Shared_I/O slaves

Entries per master system:

- Always 1 buffer with the length of 244 bytes for zeroing the inputs for CLEAR_DATA

- Always 1 buffer with the length of 244 bytes for slave addressing

- Always 1 buffer with the length of 6 bytes

Checking the calculated buffer lengths:

The total length of the buffers must be checked for the following 2 data areas.

Dual Port RAM for process data:

Length of the required buffers
= sum of the input data lengths
+ sum of the output data lengths
+ sum of the diagnostic data lengths

The individual lengths are rounded off to the next higher even number.

When consistency identifiers have been assigned on a slave (length > 1 byte), the following additional buffer lengths are required for dual port RAM.

(Over all slots)
+ length of the consistent input portion
+ length of the consistent output portion
+ filler bytes for input portion and output portion, calculated as follows:

1 byte if odd-numbered length and even-numbered offset in telegram or
if odd-numbered length and odd-numbered offset

2 bytes if even-numbered length and odd-numbered offset of identifier in telegram

0 bytes if even-numbered length and even-numbered offset

The required buffers may not exceed an upper limit of 13684 bytes in the dual port RAM.
The limit can be set via master type file (SizeRAM1).
The length of the required buffers may not exceed the upper limit. Otherwise a warning is issued.

Internal data area (IM 308-C):

Length of the required buffers
= length1 x number1 + length2 x number2 + length3 x number3 + length4 x number4

The size of the internal RAM area can be specified with the master type file (SizeRAM2).
The following upper limit applies to the buffers required in the internal RAM area.

Upper limit = internal size of 96 kbytes - dummy size

EXAMPLE of calculating the dummy size:

Size	**[in Bytes]**	**Number**	**Required Length**	**Number of Dummies**	**Dummy Size**
Buffer 1	32	3	96	0.005 ~ 1	32
Buffer 2	128	0	+ 0	0	+ 0
Buffer 3	186	0	+ 0	0	+ 0
Buffer 4	244	77	+ 18788	1.146 ~ 2	+ 488
------	------	------	= 18884	------	= 520

Number of dummies = required length / 16 Kbytes
The number of dummies is rounded off to the next higher even number.

IMPORTANT:
The number and size of the dummies is <u>not</u> entered for the required buffers.

The length of the required buffers may not exceed the upper limit calculated. Otherwise a warning is issued.

6.5.1.2.2 L4 Header Lengths/Number of APBs

General:

- The parameters " L4_Header_Len“ and “APB_Number" are not used for the IM 308-C.
- The parameters " L4_Header_Len“ and “APB_Number" are required for AMPRO2 memory management.
- The L4_Header_Längen lengths are divided into 4 categories.
- APB_Number is preset to a default value of 0 for each L4_Header_Länge.

Length	in [Bytes]	Number
L4_Header_Len1	0	0 to N
L4_Header_Len2	2	0 to N
L4_Header_Len3	4	0 to N
L4_Header_Len4	6	0 to N

L4_Header_Len must be an even number.

Range of L4_Header_Len: 0 to 32

Calculation of APB_Number depending on the master type:

APB_Number is calculated for each L4_Header_Len based on the following conditions.

APB_Number1: Number of slaves x 7

APB_Number3: Number of slaves

6.5.2 Slave Parameter Record

The following applies to all slave parameters.

The length specifications for the individual parameter data blocks (i.e., Prm_Data, Cfg_Data and so on) do not include filler bytes.

6.5.2.1 General Slave Data (Slave_Para_Data)

Address Offset BYTE	Designation	Contents	Value Range
0 to 1	*Slave_Para_Len*	*Length of the total slave parameter record including Slave_Para_Len*	*Unsigned16*
2	Sl_Flag	Status flag: 0 = Deactivated 128 = Activated	Unsigned8
3	Slave_Type	0 to 255	Unsigned8
4 to 15	Reserved		Unsigned8

[Byteoffset 0...] Slave_Para_Len:

The length of the total slave parameter record including Slave_Para_Len includes the filler bytes required when parameter data records must start at even-numbered addresses.

[Byteoffset 2] Sl_Flag:

Current status of the slave on the bus. See also ident area.

[Byteoffset 3] Slave_Type:

The slave type in accordance with the type file

Coding:	Description:
0	Standard DP slave
1 to 15	Reserved
16 ...	Manufacturer-related definitions

Slave_Type:	Allocation in Acc. w. - DP standard - SPC - SPM	Code:
DP slave SPC3	DP standard SPC	**0**
LSPM2 (ET200/B/C/DP due to ASIC)	DP standard	16
DP S7 slave	DP standard	17
ET 200U "old Siemens" IM 318-B	SPC	128
ET 200B with diagnosis (old)	SPM_Code_2	129
ET 200B without diagnosis (old)	SPM_Code_2	130
SPM - general code 0	SPM	131
SPM - general code 1	SPM	132
SPM - general code 2 ET 200C " old Siemens "	SPM	133
SPM - general code 3 ET 200K IM 418-B	SPM	134

The following method is used to distinguish between standard DP slaves and non standard DP slaves.
- Standard DP slaves are coded in the area from 0 to 127.
- Non standard DP slaves are coded in the area starting at 128.

Criteria for distinction:

MSB (most significant bit) = 0 Standard DP slave
MSB = 1 Non standard DP slave

6.5.2.2 Parameterization Data (Prm_Data)

The data which are transferred in the parameterization telegram are located here. For standard DP slaves, these data conform to the DP standard. For all others, these data conform to communication specifications V4.0 (/8/)
The parameterization data consist of bus and DP slave-related data.
In Prm_Data, a distinction must be made between standard DP slaves, SPC slaves and SPM slaves. Prm_Data has a maximum length of 245 bytes.

Caution: Prm_Data must start at an even-numbered address.
A filler byte must be added if necessary.

6.5.2.2.1 Prm_Data for Standard DP Slaves

Address Offset BYTE	**Designation**	**Contents**	**Value Range**
0 to 1 [Standard]	*Prm_Data_Len*	*Length of Prm_Data including Prm_Data_Len*	*Unsigned16*
2 [1]	station_status	7 Lock_Req 6 Unlock_Req 5 Sync_Req 4 Freeze_Req 3 Watchdog_On (WD) 2 to 0 Reserved	Unsigned8
3 [2]	WD_Factor_1		Unsigned8
4 [3]	WD_Factor_2		Unsigned8
5 [4]	TSDRmin	Station delay responder time	Unsigned8
6 [5]	Ident_Number_High	In acc. w. PNO	Unsigned8
7 [6]	Ident_Number_Low		Unsigned8
8 [7]	Group_Ident	Group	Unsigned8
9 to 244 [8 to 244]	**UserPar**	**Manufacturer-related parameterization data**	**Unsigned8**

PNO : Profibus-Nutzer-Organisation (i.e., PROFIBUS user organization)

[Byteoffset 8] Group_Ident:

Bit positions 0 to 7 correspond to groups 1 to 8.

Parameterization of standard DP slaves with the SPC3, LSPM2, SPM2 ASICs:

All station types which use one of the SPC3, LSPM2 or SPM2 ASICs require uniform parameterization provided by COM over the entire bus.
Layout of the parameterization telegrams for the affected stations types:

Attention: The first user-related byte (i.e., no. 8) is common to all types of ASICs.

6.5.2.2.1.1 Parameterization of Standard DP Slaves with the SPM2 ASIC

Address Offset BYTE		Designation	Contents	Value Range
9	[8]		7 TBD2 6 TBD1 5 DDB_EN 4 Mask_Clear_DDB 3 EN_Sammel_Dia 2 WatchDogTime_Base 1 Disable_Stopbit 0 Disable_Sartbit Default 0x00	Unsigned8
10	[9]		Mask for diagnostic port 1	Unsigned8
11	[10]		Mask for diagnostic port 2	Unsigned8
12	[11]		Mask for diagnostic port 3	Unsigned8
13	[12]		Mask for diagnostic port 4	Unsigned8

Either 8 or 12 bytes are permitted.
Length: == 8 bytes User parameters are set to their default values.
Length: != 12 bytes User parameters are used.

6.5.2.2.1.2 Parameterization of Standard DP Slaves with the LSPM2 ASIC

Address Offset BYTE		Designation	Contents	Value Range
9	[8]		7 TBD3 6 TBD2 5 TBD1 4 Mask_Clear_DDB 3 EN_Sammel_Dia 2 WatchDogTime_Base 1 Disable_Stopbit 0 Disable_Sartbit Default 0x00	Unsigned8
10	[9]		Mask for diagnostic port 1	Unsigned8
11	[10]		Mask for diagnostic port 2	Unsigned8
12	[11]		User_Def_PRM_3	Unsigned8
13	[12]		User_Def_PRM_4	Unsigned8

Only 12 bytes are permitted.
Length: == 12 bytes User parameters are used.
Length: != 12 bytes Telegram is rejected with RS.

6.5.2.2.1.3 Parameterization of Standard DP Slaves with the SPC3 ASIC

Address Offset BYTE	Designation	Contents	Value Range
9 [8]		7 TBD3 6 TBD2 5 DP_DDB_EN 4 Mask_Clear_DDB 3 TBD1 2 WatchDogTime_Base 1 Disable_Stopbit 0 Disable_Sartbit Default 0x00	Unsigned8

7 or more bytes are permitted.
Byte 8: User parameter are set to their default values.
Starting a byte 9: User parameters are used.

6.5.2.2.1.4 Parameterization of Standard DP Slaves with the ASPC2 ASIC

Address Offset BYTE	Designation	Contents	Value Range
9 [8]		Bit 7 Protocol mode (DP/DPX) 6 Fail_Safe mode 5 DP_STS 4 CLEAR_STS 3 TBD1 2 WatchDogTime_Base 1 Disable_Stopbit 0 Disable_Startbit Default 0x00	Unsigned8

7 or more bytes are permitted.
Byte 8: User parameters are set to their default values.
Starting at byte 9: User parameters are used.

6.5.2.2.2 Prm_Data for SPC Slaves

Address Offset BYTE	Designation	Contents	Value Range
0 to 1 [SPC]	*Prm_Data_Len*	*Length of Prm_Data including Prm_Data_Len*	*Unsigned16*
2 [1]	station_status	Bit position: 7 Disable_Port_Write 6 Direct_Diagnose 5 Diagnosis_Error_Off 4 AG100_Slow_Mode 3 Watchdog_On (WD) 2 to 0 Station type	Unsigned8
3 [2]	WD_Factor_1		Unsigned8
4 [3]	WD_Factor_2		Unsigned8
5 [4]	TSDRmin	Station delay responder time	Unsigned8
6 to 245 [5 to 244]	UserPar	Manufacturer-related parameterization data	Unsigned8

[Byteoffset 2] station_status:

Bit 6 / Direct_Diagnose 0: Do not add diagnosis.
1: Add diagnosis.

6.5.2.2.3 Prm_Data for SPM Slaves

Address Offset BYTE	Designation	Contents	Value Range
0 to 1 [SPM]	*Prm_Data_Len*	*Length of Prm_Data including Prm_Data_Len*	*Unsigned16*
2 [1]	Port 1		Unsigned8
3 [2]	Port 2		Unsigned8
4 [3]	Port 3		Unsigned8
5 [4]	Port 4		Unsigned8
6 [5]	Configuration	Bit position: 7 Output ports disabled/enabled 6 XDirectDiagnose 5 Mask out diagnosis 4 Trdy 1 = 55 bits (default) Trdy 0 = 11 bits 3 to 0 Data length of the response telegram	Unsigned8
7 [6]	WD_Factor_1	0x00 Watchdog_Off 0x01 to 0xFF Watchdog_Time	Unsigned8
8 [7]	This_Station	7 Store data in E2PROM 6 to 0 Station address	Unsigned8
9 [8]	Diagnostic mask 1		Unsigned8
10 [9]	Diagnostic mask 2		Unsigned8
11 [10]	Diagnostic mask 3		Unsigned8
12 [11]	Diagnostic mask 4		Unsigned8
13 to 245 [12 to 244]	UserPar	Manufacturer-related parameterization data	Unsigned8

The following applies to the ports and diagnostic masks.

	ET 200K	General SPMs
Port 1	Port A - don't care	Port A
Port 2	Port B	Port B
Port 3	Port - don't care	Port C
Port 4	Port - don't care	Port D
Diagnostic mask 1	Diag_Mask_Port_C	Diag_Mask_Port_A
Diagnostic mask 2	Diag_Mask_Port_D	Diag_Mask_Port_B
Diagnostic mask 3	Diag_Mask_Port_A	Diag_Mask_Port_C
Diagnostic mask 4	Diag_Mask_Port_BE	Diag_Mask_Port_D

[Byteoffset 6] configuration:

Bit 6 / XDirect_Diagnose 0: Add diagnosis.
1: Do not add diagnosis.

6.5.2.2.4 Triggering Monitor Time (Watchdog)

DPN slaves:

[Byteoffset 3] WD_Factor_1:
[Byteoffset 4] WD_Factor_2:

Formula: WD_time [msec]= WD_Base x WD_Factor_1 x WD_Factor_2

WD_Base in [msec]	10	1
WD_Base in byte 8 for special ASICs	0	1

WD_Base: Default: 0 / 10 [msec]
General value range for Factor_1/_2: 1 to 255
Restriction: Both factors may not be 1.

The following applies to station types with special ASICs SPC3, LSPM2 and SPM2.

- For baud rates equal to or greater than **3 Mbaud** *and* **repeater on the bus**, the DisableStartBit in parameterization byte 8 is set to 1 (default: 0).

- For baud rates equal to or greater than 1.5 Mbaud and a watchdog time of less than 40 [msec], the WatchDogTime factors are calculated again and WatchDogTime_Base is switched to 1 (i.e., to 1 [msec]).

SPC slaves:

[Byte offset 3] WD_Factor_1:
[Byte offset 4] WD_Factor_2:

Formula: WD_time [msec] = WD_Base x WD_Factor_1 x WD_Factor_2

WD_Base = 33 [msec]
General value range for Factor_1/_2: 1 to 255
Restriction: Both factors may not be 1.

SPM slaves:

[Byteoffset 7] WD_Factor_1:

Formula: WD_time [sec] = Tbit[sec] x WD_Factor_1 x WD_Factor_2

Tbit[sec] = 1 / baud rate
WD_Factor_2 = 64 for baud rates of 9.6 and 19.2 kbaud
WD_Factor_2 = 512 for baud rates greater than 19.2 kbaud
Value range for Factor_1: 1 to 255

Baud rate [kBd]:	9.6	19.2	93.75	187.5	500	1500
Max. time [sec]:	1.7	0.8	1.4	0.7	0.2	0.1

Rules for handling triggering monitor times:

The SPM type ASIC will not be handled separately here.
The WatchDog value depends on the baud rate.

Basic rules:

- An attempt should be made to keep the configured times the same for all slaves.
- The calculated time must be equal to or greater than the specified time.
- If the specified value is not within the possible value range of the factors, the minimum or maximum possible value is set as the default.

Rule 1:
Compliance with this rule is simple when only one type of ASIC is used on the bus. Use the time indicated or entered as user-defined for the export function, and round off to the next higher unit.

Rule 2:
When a bus setup using "mixed" ASICs with a combination of 1 msec and 10 msec or 1 msec and 33 msec as WatchDogTimeBase is used, all watch dog times must be set to the "high" value resulting from the 10-msec or 33-msec time base.

Rule 3:
When the bus setup includes ASICs with a WatchDogBaseTime of 1 msec, 10 msec and 33 msec or 10 msec and 33 msec (a rare situation), the next tens unit of the entered time is used. For stations with the 33 msec time base, the next value equal to or greater than the specified value is used.

6.5.2.3 Configuration Data (Cfg_Data)

The data which are transferred in the configuration telegram are located here.
For DP slaves, these data conform to the DP standard. For all others, these data conform to communication specifications.

Caution: Cfg_Data must start at an even-numbered address.
If necessary, a filler byte must be added.

The configuration data describe the real setup of the slave (i.e., slave configuration) or its modules (i.e., slots). Each slot is specified by an identifier.

The order of the entries corresponds to the slave configuration and starts with slot 0.

- "Hidden slot"

  A hidden slot was configured (i.e., reserved) but does not exist in reality.
  The master treats it as an empty slot.
  It is entered in Cfg_Data as "empty slot."
  No entry is made in the address table.
  The configuration information is stored in the private COM data to obtain the complete configuration of the master system during import.

6.5.2.3.1 Cfg_Data for Standard DP Slaves

The following identifier bytes can be entered as configuration data.

- Normal identifier byte/empty slot

  Or
- Special identifier byte as empty slot with/without commentary

  Or
- Special identifier byte with subidentifier for input with/without commentary

  Or
- Special identifier byte with subidentifier for output with/without commentary

  Or
- Special identifier byte with subidentifiers for output and subidentifier for input with/without commentary

Address Offset BYTE	Designation	Contents	Value Range
0 to 1	*Cfg_Data_Len*	*Length of Cfg_Data including Cfg_Data_Len*	*Unsigned16*
L		Normal identifier byte/empty slot	Unsigned8
	... or ...		
M		Special identifier byte as empty slot	Unsigned8
M+1...	Commentary	Manufacturer-related data	Octet-String(1 to 15)
	... or ...		
N		Special identifier byte	Unsigned8
N+1		Subidentifier for inputs	Unsigned8
N+2...	Commentary	Manufacturer-related data	Octet-String(1 to 15)
	... or ...		
O		Special identifier byte	Unsigned8
O+1		Subidentifier for outputs	Unsigned8
O+2...	Commentary	Manufacturer-related data	Octet-String(1 to 15)
	... or ...		
P		Special identifier byte	Unsigned8
P+1		Subidentifier for outputs	Unsigned8
P+2		Subidentifier for inputs	Unsigned8
P+3...	Commentary	Manufacturer-related data	Octet-String(1 to 15)
... 245			

The layout of the identifier bytes conforms to DP standard /7/.
Specifications of consistency refer to the slots. Length specifications in bytes.

Normal identifier byte:

Bit	Designation	Code	Explanation
7	Consistency	0 / 1	Byte or word/total length
6	Format	0 / 1	Byte structure/word structure
5, 4	Input/output	00 01 10 11	Special identifier format Input Output Input/output
3, 2, 1, 0	Length of the data	00 ...	1 byte/word to 16 bytes/words

Special identifier byte:

Bit	Designation	Code	Explanation
7, 6	Next byte	00 01 10 11	Empty slot A length byte for inputs follows. A length byte for outputs follows. A length byte for outputs and a length byte for inputs follow.
5, 4	Input/output	00	Fixed for special identifier format
3, 2, 1, 0	Length of the data	00 ...	Length of manufacturer-related data (commentary)

Next/subidentifier of the special identifier byte (length byte) for inputs and outputs:

Bit	Designation	Code	Explanation
7	Consistency	0 / 1	Byte or word/total length
6	Format	0 / 1	Byte structure/word structure
5 to 0	Length of the data	00 ...	1 byte/word to 64 bytes/words

Based on the layout of the identifier bytes, the following applies to slot consistency.

Slot Consistency	Format Bit 6	Consistency Bit 7
Byte consistency	0	0
Word consistency	1	0
Byte-block consistency	0	1
Word-block consistency	1	1

6.5.2.3.2 Cfg_Data for SPC Slaves

Address Offset BYTE	Designation	Contents	Value Range
0 to 1	*Cfg_Data_Len*	*Length of Cfg_Data including Cfg_Data_Len*	*Unsigned16*
2		Normal identifier byte/empty slot	Unsigned8

Layout of the identifier byte:

Bit	Code	Explanation
7	0 / 1	Write consistency, Y/N
6	0 / 1	Read consistency, Y/N
5	0 / 1	Expanded diagnosis for DI/DO, Y/N
4, 3	 00 01 10 11	Input/output: Empty slot Input Output Input/output
2	 0 / 1	Analog/digital: 0 Length nibble; consistency byte 1 Length word; consistency word
1, 0	 00 01 10 11	Length in nibbles/words 1 2 3 4

Based on the layout of the identifier bytes, the following applies to slot consistency.

Slot Consistency	Write Consistency Bit 7	Read Consistency Bit 6	Output/Input Bit 4/3	Analog/ Digital Bit 2
Byte consistency	0	0	11	0
Byte block for input	1	1	11	0
Byte block for output	0	1	11	0
Byte block for input/output	1	1	11	0

Differentiation between input and output consistency is possible.

6.5.2.3.3 Cfg_Data for SPM Slaves

Address Offset BYTE	Designation	Contents	Value Range
0 to 1	*Cfg_Data_Len*	*Length of Cfg_Data including Cfg_Data_Len*	*Unsigned16*

Poor SPM slave. No Cfg_Data for you.

Consistency for SPM slaves is fixed at '11'.

6.5.2.4 Address Table (Add_Tab)

The address allocation list of the slaves is stored here.

Address Offset BYTE	Designation	Contents	Value Range
0 to 1	*Add_Tab_Len*	*Length of total Add_Tab including Add_Tab_Len*	*Unsigned16*

Address entries for allocated input and output addresses:

2 to 3	*Input_Len*	*Length of input address entries including Input_Len*	*Unsigned16*
4...		Address entries 1 to 245	Unsigned16

M	*Output_Len*	*Length of output address entries including Output_Len*	*Unsigned16*
M+2...		Address entries 1 to 245	Unsigned16

Address entries for addressing in the DP window (dual-port RAM) [IM: Interface Module]

N1	*IM_Input_Len*	*Length of IM input address entries including IM_Input_Len*	*Unsigned16*
N+2...		Address entries 1 to 245	Unsigned16

O	*IM_Output_Len*	*Length of IM_Output address entries including IM_Output_Len*	*Unsigned16*
O+2		Address entries 1 to 245	Unsigned16

GAP address entries:
- Only for PLC 115U/CPU 941-944 and linear addressing P000 - P127

P	*GAP_Input_Len*	*Length of GAP_Input address entries including GAP_Input_Len*	*Unsigned16*
P+2...		Address entries 1 to 245	Unsigned16

Q	*GAP_Output_Len*	*Length of GAP_Output address entries including GAP_Output_Len*	*Unsigned16*
Q+2		Address entries 1 to 245	Unsigned16

6.5.2.4.1 Layout of the Address Entries

Bits 15 - 14	**Bit 13**	**Bit 12**	**Bits 11 - 8**	**Bits 7 - 0**
Consistency information	Page frame addressing	P/Q area	Page frame number	Byte offset of the S5 address

- Consistency information:

Only "short" consistency is identified in the consistency bits.

15	**14**	**Meaning**	**Consistency**	**RDY**	**Consistency Control**
0	1	Beginning of consistent area	Beginning	Yes	On
1	1	End of consistent area	End	Yes	Off
1	0	Consistent area, slave failure	End	No	Off
0	0	Address not allocated	Disregard	No	Unchanged

- Page frame addressing: Page frame Y = 1 | N = 0

- P/Q area: P area = 0 | Q = 1

- Page frame number: Page frame number contains coding to 4 bits of offset to the base page frame.

- Byte offset: The field codes the S5 address in the selected area.

Example of page frame addressing and the P/Q area:

Contents	Description
'00'	P linear
'01'	Q linear
'10'	P page frame
'11'	Q page frame

6.5.2.4.2 Consistency Information of the Address Entries

Address entries are slot-related depending on the consistency.

- BYTE consistency (i.e., no consistency)
- WORD consistency
- BYTE-BLOCK consistency
- WORD-BLOCK consistency

6.5.2.4.3 Special Characteristics of the Address Entry

Slave type ET 200U:

Definition in accordance with general slave data (Slave_Para_Data):
ET 200U "old Siemens" IM 318-B SPC

The entry is made in the following order.

1. Addresses of analog modules (slots)
   Word entries must be stored in Intel format (i.e., converted).
2. Addresses of digital modules

6.5.2.5 User Data (Slave_User_Data)

Address Offset BYTE	Designation	Contents	Value Range
0 to 1	*Slave_User_Data_Len*	*Length of Slave_User_Data including Slave_User_Data_Len*	*Unsigned16*
2	SAP_Chk_Cfg	= 62	Unsigned8
3	SAP_Set_Prm	= 61	Unsigned8
4	SAP_Slave_Diag	= 60	Unsigned8
5	SAP_Get_Cfg	= 59	Unsigned8
6	SAP_Global_Control	= 58	Unsigned8
7	SAP_RD_Outp	= 57	Unsigned8
8	SAP_RD_Inp	= 56	Unsigned8
9	SAP_Set_Slave_Add	= 55	Unsigned8
10	SAP_DST	= 53	Unsigned8
11	SAP_Special_Fct1	255 = don´t care (cf. Prm_Data)	Unsigned8
12	SAP_Special_Fct2	255 = don´t care (cf. Prm_Data)	Unsigned8
13	Ident_Number_High	In acc. w. PNO (cf. Prm_Data)	Unsigned8
14	Ident_Number_Low		Unsigned8
15	Fail_Save_Modus	Fail-safe capability of a slave: FALSE | TRUE (from type file) Default: FALSE	Unsigned8
16	SPM_HWIdent_High		Unsigned8
17	SPM_HWIdent_Low		Unsigned8
18	DiagRespLen	Length of the diagnostic telegram for all slaves	Unsigned8
19	OutputDataLen	Length of output data used	Unsigned8
20	InputDataLen	Length of input data used	Unsigned8
21	DiagDataLen	Max. length of the diagnostic data after conversion in the dual-port RAM	Unsigned8
22	SlaveChnLen	Max. length of the slave channel (0 since not used at this time)	Unsigned8
23	Error mode	0 No error mode	Unsigned8
24	Consistency		Unsigned8
25	Slave operating mode		Unsigned8
26...	Flags		Unsigned32
30		Reserved	Unsigned8

[Offset 18] Length of the diagnostic telegram for all slaves:
[Offset 21] Max. length of the diagnostic data after conversion in the dual-port RAM:

The length of the diagnostic data [in bytes] is calculated as shown below.

Slave Allocation	[18] Length of the Telegram	[21] Max. Length of the Data
Standard DP slaves	6 to 244	6 to 244
SPC slaves	6 7 to 32	13 fixed 15 to 40 (Offset of 8 bytes)
SPM slaves	11 - SPM type (SPM type = number)	13 fixed
Own definition	See standard DP slaves.	See standard DP slaves.

[Offset 24] Consistency:

Consistency differentiates between inputs and outputs.

Type	Bit	Description
Inputs	7	0 = Not buffered / 1= Buffered
	6, 5	00 = No consistency 01 = "Short" consistency 10 = "Long" consistency 11 = Reserved
	4	= 0
Outputs	3	0 = Not buffered / 1= Buffered
	2, 1	00 = No consistency 01 = "Short" consistency 10 = "Long" consistency 11 = Reserved
	0	= 0

Table for determining the type of addressing and information on consistency and buffering for the parameter module:

	No Consistency			**With Consistency**		
Applies to **inputs/outputs**	**I**		**O**	**I/O**		
Total telegram length in bytes	≤ 122*	> 122*	≤ 244*	≤ 122*		> 122*
Longest consistent area of an identifier in bytes (station-wide)	1	1	1	1 < area ≤ CB**	> CB**	> 1
Parameter module: **Consistency** None/short/long	None	None	None	Short	Long***	Long***
Parameter module: **Buffering** Yes/no	No	Yes	No	No	Yes	Yes
Implementation in IM 308-C stage	V 5.0	V 5.2	V 5.0	V 5.0	V 5.2	V 5.2

* The limits which are listed below are entered in the master type file.

Longest Consistent Area of an Identifier in Bytes (Station-Wide)	**Inputs**	**Outputs**
1	122	244
1 < area ≤ consistency boundary	122	122

CB** Boundary between "long" and "short" consistency

*** When "long" consistency is used, all address pointers must be entered in the parameter module with a consistency of "11" (i.e., last consistent address).

Remark: A slave can be operating with "long" consistency for the inputs and "short" consistency for the outputs.

[Offset 25] Slave operating mode:

Bit Position	**Significance**	**Designation**
7 to 3	---	Reserved
2	0/1	Read_Output
1	0/1	Read_Input
0	0/1	XchangeData

Bit 0	**Bit 1**	**Bit 2**	**Possible Settings**	
1	0	0	XChangeData	
0	1	0	Read_Input	
0	0	1	Read_Output	Not yet available
0	1	1	Read_Input / Output	Not yet available

[Offset 26] Flags

Slave information from the type file (binary-coded):

0 Slave is a Siemens slave.
1 Slave is a standard DP slave.
2 Slave can be programmed via bus.
3 Slave can be synced.
4 Slave can be frozen.
5 Slave is an ASI gateway in default mode.
6 Slave can buffer diagnosis.
7 Slave is active.
8 Slave requires even slot number.
9 Slave is an ASI gateway.
10 Slave is also a master.
11 Slave is publisher.
12 Slave is subscriber.
13 Only normal identifier format is permitted for slave.
14 Only special identifier format is permitted for slave.
15 Slave is S7 slave with 2 permanently preconfigured slots.
16 Slave is S7 slave with 3 permanently preconfigured slots.
17 Slave has I/O with channel capability.

7 Appendix

7.1 Address List

PROFIBUS Nutzer Organisation
PNO
Office
Mr. Dr. Peter Wenzel
Haid- und Neu- Strasse 7
76131 Karlsruhe/Germany
Tel.: (0721) 9658-590

Contact Persons at the Interface Center in Germany

Siemens AG
Deptl IA SE DE DP3
Mr. Putschky
Würzburgerstr.121
90766 Fürth/Germany

Email:
gerd.putschky@siemens.com

Tel.: (0911) 750 - 2078
Fax: (0911) 750 - 2100

Contact Persons at the Interface Center in the USA

PROFIBUS Interface Center
One Internet Plaza
PO Box 4991
Johnson City, TN 37602-4991

Fax : (423) - 262 - 2103

Your Partner:
Tel.: (423) - 262 - 2576

Email:
profibus.sea@siemens.com

7.2 List of Related Literature

/1/ PROFIBUS-Norm DIN 19245, Teil 1. April 1991.

/2/ PROFIBUS-Norm DIN 19245, Teil 3. Normentwurf der PNO (PROFIBUS-Nutzerorganisation), Juli 1994.

/3/ Kiendl, M.: Terminologie-Datenbank für ET200, Version V 2.2. Siemens AG Amberg, Abt. AUT 125, 07.04.1995.

/4/ Gebhardt, M.: Parametermodul für ET200-Kernsystem V 5.0, Version V 1.04. Dokumenten-Nummer ET2_S010, Siemens AG Erlangen, Abt. AUT 624, 04.05.1995.

/5/ Heinrich, A. / Spichtinger, K. / Tretter, B.: AMPRO2 User-Interface, Version V 2.4. Siemens AG Amberg, Abt. AUT 1242B, 20.10.95 zur AMPRO2-Firmware V 2.4 vom 20.10.95.

/6/ Spichtinger, K. / Tretter, B.: AMPRO2 Application-Notes, Version V 2.3. Siemens AG Amberg, Abt. AUT 1242B, 18.07.95.

/7/ Steindl, G.: Rahmen-Spezifikation, Version V 1.3. Siemens AG Amberg, Abt. AUT 1242B, 21.09.94.

/8/ Steindl, G.: Kommunikationsspezifikation ET 200, Version V 4.0. Siemens AG Amberg, Abt. AUT 1242B, Januar 1994.

7.3 List of Abbreviations

Most of the abbreviations used in these specifications are explained the terminology data base. Most of the abbreviations listed in this section are not included in the data base.

AMPRO	**A**dvanced **M**ulti-user **PRO**FIBUS
APB	**Ap**plication **B**lock
ASPC2	**A**dvanced **S**iemens-**P**ROFIBUS **C**ontroller **2**; ASIC
AST	**A**UTO**ST**OP; master operating mode
B	Behind numbers; number system based on 2 (**B**inary)
CB	**C**ontrol **B**lock
CBF	**C**all **B**ack **F**unction
CFG	**C**on**fig**uration slave state
D	Behind numbers; number system based on 10 (**D**ecimal)
DATA	**Data** exchange slave state; data communication
DATA_NA	**Data** Exchange but **N**ot **A**vailable slave state
DATR	**Da**ta **Tr**ansfer
DB	**D**ata **Bl**ock
DEACT	**Deact**ivated slave state
DIAG1	**Diag**nosis **1** slave state
DIAG2	**Diag**nosis **2** slave state
DP	**D**ecentral **P**eriphery
DPM	**DP**-**M**aster; communication functions of a class-1 DP master in acc. w. standard
DPMIB	**DPM I**nit **B**lock
DPS	**DP S**lave; communication functions of a DP slave in acc. w. standard
DTL	**D**ata **T**ransfer **L**ist
ERRCB	**Err**or **C**ontrol **B**lock
FASA	**Fa**il**sa**fe operating mode
FDL	**F**ieldbus **D**ata **L**ink
FLC	**F**ieldbus **L**ink **C**ontrol
FMA	**F**ieldbus **Ma**nagement Layer
FW	**F**irm**w**are
H	Behind numbers; number system based on 16 (**H**exadecimal)
HW	**H**ard**w**are
IFA	**I**nter**fa**ce
ISO	**I**nternational **O**rganization for **S**tandardization
L	Direction of Access **L**esen (lesen = read)
L2	**L**ayer **2** of the ISO/OSI 7 layer model
L4	**L**ayer **4** of the ISO/OSI 7 layer model
LSB	**L**east **S**ignificant **B**it
MAC	**M**edium **A**ccess **C**ontrol

MARKCB	**mark**_cycle **C**ontrol **B**lock
MSB	**M**ost **S**ignificant **B**it
OSI	**O**pen **S**ystem **I**nterconnection
PDA	**P**rocess **D**ata **A**rea; process data area
PRM	**P**ara**m**eterization of slave state
PRM_UNLOCK	Cancellation of **P**ara**m**eterization of slave state
PROFIBUS	**Pro**cess **Fi**eld**bus**
Req	**Req**uest
Res	**Res**ponse
S	Direction of Access **S**chreiben (schreiben = write)
SCB	**S**ystem **C**ontrol **B**lock for memory area required for the ASPC2
SD	**S**ystem **D**iagnosis
SDA	**S**end **D**ata with **A**cknowledge, PROFIBUS service
SDL	**S**pecification and **D**escription **L**anguage
SDN	**S**end **D**ata with **N**o Acknowledge; PROFIBUS service
SIO_DIAG2	**S**hared_**I**nput/**O**utput slave state **Diag**nose **2**
SIO_GET_CFG	**S**hared_**I**nput/**O**utput slave state **Get_C**on**fig**
SIO_RD_IO	**S**hared_**I**nput/**O**utput slave state **R**ea**d_I**nput/**O**utput
SLCB	**Sl**ave **C**ontrol **B**lock
SLSM	**Sl**ave **S**tate **M**achine
SMMCB	**s**et_**m**aster_**m**ode **C**ontrol **B**lock
SPC	**S**iemens **P**ROFIBUS **C**ontroller; ASIC
SPC	**S**iemens **P**ROFIBUS **M**ultiplexer; ASIC
SRD	**S**end and **R**equest **D**ata with reply; PROFIBUS service
SSLACB	**s**et_**sl**ave_**a**ddress **C**ontrol **B**lock
SSMCB	**s**et_**s**lave_**m**ode **C**ontrol **B**lock
USIF	**Us**er **I**nter**f**ace