The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC871-SW-42

VxWorks Device Driver

TPMC871 – 1 Slot PC Card Interface PMC
TPMC872 – 2 Slot PC Card Interface PMC
TCP872 – 2 Slot PC Card Interface cPCI

Version 2.5.0

User Manual

Issue 2.5

September 2004

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7
25469 Halstenbek / Germany
Phone: +49-(0)4101-4058-0
Fax: +49-(0)4101-4058-19
e-mail: info@tews.com
www.tews.com

TEWS TECHNOLOGIES LLC
1 E. Liberty Street, Sixth Floor
Reno, Nevada 89504 / USA
Phone: +1 (775) 686 6077
Fax: +1 (775) 686 6024
e-mail: usasales@tews.com
www.tews.com

TPMC871-SW-42

PC Card Interface PMC

VxWorks Device Driver with ATA Disk Driver

Issue	Description	Date
2.0	First Issue	July 2000
2.1	Extended Configuration Description	August 2001
2.2	General Revision	January 2004
2.3	Errors in description corrected	March 2004
2.4	TPMC871 V3 and TPMC872 are now supported	March 2004
2.5	Installation and Setup modified, TCP872 is now supported	September 8, 2004

Table of Content

1 INTRODUCTION 5
2 INSTALLATION 6
2.1 Before compiling 7
2.1.1 Hardware and BSP Adaptations 7
2.1.2 Include PCI setup for TPMC871 8
2.1.3 Setup Number of Used Adapters 8
3 PC CARD INTERFACE FUNCTIONS 9
3.1 Dependent Functions 9
3.1.1 tpmc871_ss_init() 9
3.1.2 tpmc871_init() 11
3.1.3 Entry Point Function 13
3.1.4 Socket Functions 15
3.2 Calling Interface Functions 16
3.2.1 pcc_init() 16
3.2.2 pcc_adapter_init() 18
3.2.3 pcc_entry() 20
4 SOCKET FUNCTIONS 22
4.1 Supported Functions 22
4.1.1 GetAdapterCount 22
4.1.2 InquireAdapter 24
4.1.3 GetAdapter 27
4.1.4 SetAdapter 29
4.1.5 InquireWindow 31
4.1.6 GetWindow 36
4.1.7 SetWindow 39
4.1.8 GetPage 43
4.1.9 Set Page 45
4.1.10 InquireSocket 47
4.1.11 GetSocket 50
4.1.12 SetSocket 55
4.1.13 GetStatus 60
4.1.14 ResetSocket 64
4.1.15 GetVendorInfo 66
4.1.16 AcknowlegdeInterrupt 68
4.2 Unsupported Functions 70
4.2.1 GetSSInfo 70
4.2.2 InquireEDC 70
4.2.3 GetEDC 71
4.2.4 SetEDC 71
4.2.5 StartEDC 72
4.2.6 PauseEDC 72
4.2.7 ResumeEDC 73
4.2.8 StopEDC 73
4.2.9 ReadEDC 74
4.2.10 GetSetPriorHandler 74
4.2.11 GetSetSSAddr 75
4.2.12 GetAccessOffsets 75
4.2.13 AccessConfigurationSpace 76
4.2.14 InquireBridgeWindow 76
4.2.15 GetBridgeWindow 77

4.2.16 SetBridgeWindow 77
4.2.17 VendorSpecific 78
5 ATA DISK DEVICE DRIVER 79
5.1 I/O System Functions 79
5.1.1 pccAtaDrv() 79
5.1.2 pccAtaDevCreate() 81
5.2 I/O Interface Functions 83
5.2.1 open() 83
5.2.2 read() 84
5.2.3 write() 85
5.2.4 ioctl() 86
6 APPLICATION HINTS 87
6.1 Start up PC Card Socket Functions 87
6.2 Start up the PC Card ATA Driver 90
6.3 Setup Access Memory Card 94
7 APPENDIX 96
7.1 Error Codes 96
7.1.1 TPMC871 Error Codes 96
7.1.2 Socket Function Error Codes 96
7.1.3 ATA Disk Driver Error Codes 97

1 Introduction

The TPMC871-SW-42 device interface software allows the operation of the following module types:

- TPMC871 1 slot PC Card Interface PMC (V1.x … V3.x)
- TPMC872 2 slot PC Card Interface PMC
- TCP872 2 slot PC Card Interface compact PCI Card

In the following description TPMC871 will be used for all covered module types. If there are differences handling the modules type, this is described in the according paragraph.

This device interface software is split into three layers:

- Hardware dependent layer
- Calling interface for hardware independent calls
- Application layer or PC Card device driver layer (PC Card ATA Disk driver)

The hardware dependent part mainly covers the functions of the socket services defined in the PC Card standard. These functions get information about the controllers setting and status as well as the capabilities of the device interface. Other functions handle the setup of the controllers. All this functions are called through one entry point with a function code and a variable number of parameters. Functions which are not defined in the PC Card standard are called directly. These functions will initialize the device interface and PC Card controllers.

The calling interface can handle calls to different PC Card interfaces. This will allow a simple change of the hardware layer or to use different hardware layers with the same PC Card device drivers.

The application and PC Card device driver layer handles the access to the PC Cards. This layer will call the hardware independent calling interface to allow access to the PC Cards.

2 Installation

The software is delivered on a 3½" HD diskette.

Following files are located on the diskette:

tp871_ss.c	Device Interface Source Codes
tp871_sys.c	Hardware dependent functions source code
tp871pci.c	Configuration Space Setup and Mapping function source code
tpxxxhwdep.c	Platform dependent interface functions
tp871_ss.h	Device interface setup and include file
tp871_err.h	TPMC871 specific error codes
tp871d_ss.h	Device interface include file
tp871_sys.h	MVME2300 dependent include file
tpxxxhwdep.h	Platform dependent include file
pcc_mng.c	Calling interface source code
pcc_mng.h	Calling interface and application include file
ata_drv.c	PC Card ATA Disk device driver source code
ata_drv.h	ATA device driver and application include file
ata_def.h	Local ATA device driver include file
pcc_test.c	Example application for PC Card interface software
ata_test.c	Example application for PC Card ATA Disk device driver
pciMapShow.c	Function showing the PCI devices and a Memory- and IO-Map
TPMC871-SW-42.pdf	VxWorks TPMC871 PC Card Device Interface Software manual (this manual)

For installation the files have to be copied to the desired target directory.

2.1 Before compiling

2.1.1 Hardware and BSP Adaptations

Before compiling the software some adaptations to the hardware and BSP have to be done.

Processor dependent adaptations must be made in *tpxxxhwdep.c*. This file supports PowerPC and Intel x86-family. If another processor family is used, check if it will run with one of the families. Otherwise a part into the functions which handles this family type has to be added.

BSP and system dependent adaptations must be made in *tp871_sys.h*, *tp871_sys.c*, and *tp871pci.c*. In *tp871_sys.h* areas have to be specified where the memory and I/O windows can be mapped to.

TP871_SUPPORTED_BUSES is defined in tp871_sys.h and defines the maximum PCI bus number. The value should be at least the PCI bus number where the TPMC871 will be found plus one. (PCI bus numbers will start with zero)

The data array *tp871_bus_config_table* must be adapted to match to your BSP and system. The array defines PCI bus number dependent values. The array index is equal to the PCI bus number. It is necessary to adapt the entries of the array where the TPMC871s are mounted to. All other entries can be left in default setup. The data structure of the array entries is defined as *TP871_BUS_CONFIG_TABLE* in *tp871_sys.h*. See the layout below.

```
typedef struct
{
        unsigned long                   memoryOffset;
        unsigned long                   ioOffset;

        unsigned long                   memRegisterStartAddr;
        unsigned long                   memRegisterSize;
        unsigned long                   memWindowStartAddr;
        unsigned long                   memWindowSize;
        unsigned long                   ioWindowStartAddr;
        unsigned long                   ioWindowSize;

        int                             intLine[32];
} TP871_BUS_CONFIG_TABLE;
```

The values of ***memoryOffset*** and ***ioOffset*** define the address offsets between CPU address and PCI address. These values are BSP dependent.

The values of ***memRegisterStartAddr***, ***memRegisterSize***, ***memWindowStartAddr***, ***memWindowSize***, ***ioWindowStartAddr***, and ***ioWindowSize*** define the PCI spaces which will be used to access the TPMC871. The specified areas must be accessible via PCI bus and not be used by any other device. ***xxxStartAddr*** defines the first byte of the space, ***xxxSize*** defines the size in bytes. (xxxStartAddr + xxxSize – 1 is the last byte of the space.) The values ***memRegisterStartAddr*** and ***memRegisterSize*** define the PCI space the TPMC871 registers will be mapped to. We will need a size of 4Kbyte per PC-Card slot. The values ***memWindowStartAddr*** and ***memWindowSize*** define the space memory windows of the PC-Cards can be mapped to. The needed size of this area depends on the PC-Card and the driver of the PC-Card (e.g. the ATA-driver will use 2 windows, one with a size of 1kByte and the second with a size of 1kByte). The values ***ioWindowStartAddr*** and ***ioWindowSize*** define the space I/O windows of the PC-Cards can be mapped to. The needed size of this area depends on the PC-Card and the driver of the PC-Card. (The ATA-driver will not use any I/O space)

The array ***intLine*** defines the value of the intLine of the PCI slots the index is the PCI device number. Information how the interrupt lines are mapped, must be taken from the manuals of the BSP and the board.

We have added a function *pciMapShow()* that will display a map of the PCI bus. This may help finding free address areas on the PCI bus. To use this function add *pciMapShow.c* into your project files.

Refer to your BSP and board documentation to find out the memory-, I/O Offsets and the interrupt mapping. There is no unified way to get this on different BSP and boards.

2.1.2 Include PCI setup for TPMC871

Most of the BSPs are not supporting the PCI header type 2 which is used for the TPMC871 and are just ignoring the TPMC871 during PCI configuration. The function *tp871PciInit()* sets up the TPMC871 PCI header. To use this function the values ***memRegisterStartAddr*** and ***memRegisterSize*** must be set to valid values. If the board has already been set up, when calling the function the configuration will not be changed.

On Intel x86 systems we have to add the new spaces to MMU initialization to allow access to these spaces. This will also be done by *tp871PciInit()*.

The C source file *tp871pci.c* contains the function *tp810PciInit().* This routine finds out the TPMC871 devices and adds MMU mapping entries for all used PCI spaces. Insert a call to this function after the PCI initialization is done and prior to MMU initialization (*usrMmuInit()*). The right place to call the function *tp810PciInit()* is at the end of the function *sysHwInit()* in *sysLib.c* (can be open from the project *Files* window).

Be sure that the function is called prior to MMU initialization otherwise the TPMC871 PCI spaces remains unmapped and we got an access fault during driver initialization.

Insert the following call at a suitable place in *sysLib.c*:

```
tp871PciInit();
```

Avoiding compilation a warning a function prototype should be defined at top of the calling function. There for insert:

```
void tp871PciInit();
```

The Function *tp871PciInit()* was designed for and tested on generic Pentium targets. If you use another BSP please refer to BSP documentation or contact the technical support for required adaptation.

If you got strange errors after system startup with the new build system please carry out a VxWorks *build clean* and *build all*.

2.1.3 Setup Number of Used Adapters

The last adaptation must be made in *tp871_ss.h*. *TPMC871_MAXADAPTER* specifies the maximum number of TPMC871 which can be handled by this device interface. To handle more the value has to be enlarged.

3 PC Card Interface Functions

These functions are needed to setup the PC Card interfaces or to get information about the setup and the capabilities.

These functions are split into two parts. The first part is hardware dependent and special for the TPMC871. The other part is a device independent interface, which allows a unified access to different kind of PC Card slots.

A big part of the hardware dependent functions are the socket functions which will be described later in the chapter “Socket Functions”. These functions are defined in the PC Card standard as the socket services.

3.1 Dependent Functions

This chapter describes the functions of the TPMC871 dependent layer. This part of the driver accomplishes the access to the PC Card controller on the TPMC871.

3.1.1 tpmc871_ss_init()

NAME

tpmc871_ss_init() initializes socket interface.

SYNOPSIS

```
unsigned long     tpmc871_ss_init
(
        void
)
```

DESCRIPTION

This function initializes the local data of the TPMC871 socket interface driver. This function must be called before any other function of the hardware dependent interface is called.

EXAMPLE

```
#include          “tp871_ss.h”

...

unsigned long     result;

...

result = tpmc871_ss_init();
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

TP871_NO_ERROR (0) = if no error occurred

Otherwise an appropriate error code (see below chapter “Error Codes”)

3.1.2 tpmc871_init()

NAME

tpmc871_init() initializes adapter and gets entry point.

SYNOPSIS

```
unsigned long     tpmc871_init
(
        int                     busNo,
        int                     devNo,
        int                     funcNo,
        ADAPTER                 *adapter,
        unsigned long           (**entry)()
)
```

DESCRIPTION

This function must be called for every adapter (TPMC871 module) before any of the socket functions is called for the adapter and after the *tpmc871_ss_init()* is called. This function allocates and initializes memory for the device control block and makes a basic setup of the PC Card controller. This function reads the PCI configuration and stores the controller access areas.

PARAMETERS

The input parameters **busNo**, **devNo** and **funcNo** specify the slot where the TPMC871 is mounted to. **funcNo** must always be zero, because there is only one function implemented on the TPMC871.

This function returns the **entry** point of the socket functions and the local **adapter** number of the specified TPMC871. Both of the parameters will be needed for calling the socket functions directly and for connecting this adapter (TPMC871) with the calling interface (see chapter “Calling Interface Functions”).

EXAMPLE

```
#include          “tp871_ss.h”
#include          “pcc_mng.h”

...

unsigned long     result;
int               busNo, devNo, funcNo;
ADAPTER           locAdapter;
unsigned long     (*locEntry)();

...

/*-------------------------------------------------------
  Initialize TPMC871 mounted to
    PCI bus 0, PCI device 16 and PCI function 0
  -------------------------------------------------------*/
result = tpmc871_init(  0, 16, 0, &locAdapter, &locEntry);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
     /* use returned parameters to call the socket functions */
}

...
```

RETURNS

TP871_NO_ERROR (0) = if no error occurred

Otherwise an appropriate error code (see below chapter “Error Codes”)

3.1.3 Entry Point Function

NAME

tpmc871_entry() Entry Point Function for socket function of the TPMC871

SYNOPSIS

```
LOCAL unsigned long     tpmc871_entry
(
        unsigned long         function,
        ...
)
```

DESCRIPTION

This function is local to the tpmc871 dependent code. The only way to call this function is to get the entry point of this function by calling the *tpmc871_init()* function and using the returned address for a referenced function call. This function will call the specified socket function with the needed number of parameters. Therefore the parameter list of this function is variable.

PARAMETERS

The parameter **function** specifies which socket function shall be called.

The length and contents of parameter list depends on the called socket function.

The length of the parameter list will not be checked by this function.

EXAMPLE

```
#include          “tp871_ss.h”
#include          “pcc_mng.h”

...

unsigned long     result;
SKTBITS           Sockets;

...

/*-------------------------------------------------------
  Call the entry point locEntry and execute the socket
  function Acknowledge Interrupt for the adapter specified
  in locAdapter. locEntry and locAdapter were returned
  by tpmc871_init
  -------------------------------------------------------*/
result = locEntry( PCC_ACK_INTERRUPT,
                   (unsigned long)locAdapter,
                   (unsigned long)&Sockets);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

TP871_NO_ERROR (0) = if no error occurred

Otherwise an appropriate error code (see below chapter “Error Codes”)

3.1.4 Socket Functions

The socket functions will be described below in a separate chapter “Socket Functions”. Not all functions are necessary for every kind of PC Card controller. Therefore some of the functions are not implemented. The following list shows the possible functions and if functions are implemented for the TPMC871 or not.

Function Code	Impl.	Function Code	Impl.
PCC_GET_ADP_CNT	Yes	PCC_GET_SS_INFO	No
PCC_INQ_ADAPTER	Yes	PCC_INQ_EDC	No
PCC_GET_ADAPTER	Yes	PCC_GET_EDC	No
PCC_SET_ADAPTER	Yes	PCC_SET_EDC	No
PCC_INQ_WINDOW	Yes	PCC_START_EDC	No
PCC_GET_WINDOW	Yes	PCC_PAUSE_EDC	No
PCC_SET_WINDOW	Yes	PCC_RESUME_EDC	No
PCC_GET_PAGE	Yes	PCC_STOP_EDC	No
PCC_SET_PAGE	Yes	PCC_READ_EDC	No
PCC_INQ_SOCKET	Yes	PCC_PRIOR_HANDLER	No
PCC_GET_SOCKET	Yes	PCC_SS_ADDR	No
PCC_SET_SOCKET	Yes	PCC_ACCESS_OFFSETS	No
PCC_GET_STATUS	Yes	PCC_ACCESS_CONFIG	No
PCC_RESET_SOCKET	Yes	PCC_INQ_BWINDOW	No
PCC_GET_VENDOR_INFO	Yes	PCC_GET_BWINDOW	No
PCC_ACK_INTERRUPT	Yes	PCC_SET_BWINDOW	No
		PCC_VEND_SPECIFIC	No

3.2 Calling Interface Functions

This chapter describes the functions of the calling interface dependent layer. This part of the software implements a calling interface which allows applications and PC Card drivers to use different PC Card controllers without changing the application or driver.

3.2.1 pcc_init()

NAME

pcc_init() initializes calling interface.

SYNOPSIS

```
unsigned long     pcc_init
(
     void
)
```

DESCRIPTION

This function initializes the local data of the calling interface. This function must be called once before any other function of the calling interface is called.

EXAMPLE

```
#include          “pcc_mng.h”

...

unsigned long     result;

...

result = pcc_init();
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR (0) = if no error occurred

Otherwise an appropriate error code (see below chapter “Error Codes”)

3.2.2 pcc_adapter_init()

NAME

pcc_adapter_init() connects an adapter to the calling interface.

SYNOPSIS

```
unsigned long     pcc_adapter_init
(
        ADAPTER           adapter,
        ADAPTER           adapter_mod,
        unsigned long     (*entry)()
)
```

DESCRIPTION

This function connects an adapter to the calling interface. This function must be called before the *pcc_entry()* function is called for the specified adapter, but behind the *pcc_adapter_init()* function.

PARAMETERS

The parameter **adapter** specifies the adapter number, which shall be used in the future calls.

The parameter **adapter_mod** specifies the local adapter number. **adapter_mod** parameter is not the same as the **adapter** parameter. To get the local adapter number the hardware dependent function *tpmcxxx_init()* has to be called.

The parameter **entry** specifies the local entry point of the hardware dependent layer. The entry point will be get, if the hardware dependent function *tpmcxxx_init()* is called.

EXAMPLE

```
#include           “pcc_mng.h”

...

unsigned long      result;

...

/*-------------------------------------------------------
  Connect an initialized adapter to the calling interface.
    The adapter shall be called as adapter 4.
    locEntry and locAdapter are values returned by
    tpmcxxx_init()
  -------------------------------------------------------*/
result = pcc_adapter_init(   4, locAdapter, locEntry);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR (0) = if no error occurred

Otherwise an appropriate error code (see below chapter “Error Codes”)

3.2.3 pcc_entry()

NAME

pcc_entry() Calling interface function

SYNOPSIS

```
unsigned long      pcc_entry
(
        unsigned long        function,
        ...
)
```

DESCRIPTION

This function will call the socket function via the local entry points of the hardware dependent interfaces.

PARAMETERS

The parameter **function** specifies which socket function shall be called.

The length and contents of parameter list depends on the called socket function.

The length of the parameter list will not be checked by this function.

EXAMPLE

```
#include          “pcc_mng.h”

...

unsigned long     result;
SKTBITS           Sockets;

...

/*-------------------------------------------------------
  Call the Acknowledge interrupt function for the connected
  adapter 4
  -------------------------------------------------------*/
result = pcc_entry (    PCC_ACK_INTERRUPT,
                        4,
                        (unsigned long)&Sockets);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR (0) = if no error occurred

Otherwise an appropriate error code (see below chapter “Error Codes”)

4 Socket Functions

This chapter will describe the socket functions which are supported by the TPMC871 device interface. The unsupported functions are only listed with a short description. If different PC Card interfaces are used, it may be possible, that there are different functions supported. Please refer also to the other software manuals.

These functions are called via the calling interface in the way described before in the chapter *pcc_entry().*

4.1 Supported Functions

4.1.1 GetAdapterCount

NAME

GetAdapterCount returns the number of activated PC Card adapter located in the system.

FUNCTION CODE

PCC_GET_ADP_CNT (0x80)

PARAMETERS

COUNT *TotalAdapters

SIGNATURE Signature

DESCRIPTION

This function identifies the PC Card socket functions and returns the number of activated PC Card adapters in the system.

PARAMETERS

The parameter **TotalAdapters** points to the count of activated PC Card adapters.

The **Signature** is a field of two characters where an ident code for the socket interfaces is returned. The value returned in signature is always *'SS'* if a socket interface is present. This field must be ‘0’ before this function is called.

EXAMPLE

```
#include          “pcc_mng.h”

...

unsigned long     result;
ADAPTER           TotalAdapters;
SIGNATURE         Signature;

...

/*-------------------------------------------------------
  Get adapter count
  -------------------------------------------------------*/
Signature[0] = Signature[1] = 0;
result = pcc_entry (    PCC_GET_ADP_CNT,
                        (unsigned long)&TotalAdapters,
                        (unsigned long)Signature);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR	no error

4.1.2 InquireAdapter

NAME

InquireAdapter Get the capabilities of the specified adapter

FUNCTION CODE

PCC_INQ_ADAPTER (0x84)

PARAMETERS

ADAPTER	Adapter
PTR	pBuffer
COUNT	*NumSockets
COUNT	*NumWindows
COUNT	*NumEDCs
COUNT	*NumBridgeWindows

DESCRIPTION

This function informs about the technical capabilities of the specified adapter. The information shows the possible interrupts, possible power settings and special capabilities (for example: power down mode).

PARAMETERS

The parameter **Adapter** specifies the PC Card adapter.

The parameter **pBuffer** is a pointer to the *AISTRUCT* structure where the adapter capabilities will be stored.

Data structure *AISTRUCT*:

WORD	wBufferLength	This is an input entry and must be set to the maximal length of the *CharTable*, *wNumPwrEntries* and the *PwrEntry* array.
WORD	wDataLength	The function will return in this entry to the real length of the *CharTable*, *wNumPwrEntries* and the *PwrEntry* array.
ACHARTBL	CharTable	This structure holds the information about the interrupt and special abilities of the PC Card controller.
WORD	wNumPwrEntries	Number of elements in the *PwrEntry* array
PWRENTRY	PwrEntry[1]	Array with *wNumPwrEntries* elements, which identifies the possible power settings and levels.

Data structure *ACHARTBL*:

FLAGS8	AdpCaps	This entry shows the abilities of the adapter. Defined flags are: PCC_AC_IND – Shared indicators PCC_AC_PWR – No individual power levels PCC_AC_DBW – Same data bus width for all windows PCC_AC_CARDBUS – All sockets are CardBus PC Card capable
BYTE	CacheLineSize	Specifies the host system cache line size in units of 32 bit words. This value is zero for non CardBus PC Card adapters.
FLAGS32	ActiveHigh	This bit field describes which interrupt routings can possibly used for high active interrupts.
FLAGS32	ActiveLow	This bit field describes which interrupt routings can possibly used for low active interrupts.

Data structure *PWRENTRY*:

PWRINDEX	PowerLevel	This entry describes the power level in 1/10V (Example: 50 means 5.0V
FLAGS8	ValidSignals	This bit field describes the power lines which can be set to the previously defined power level. Defined flags are: PCC_VCC – Power index is valid for Vcc PCC_VPP1 – Power index is valid for Vpp1 PCC_VPP2 – Power index is valid for Vpp2

The parameter **NumSockets** returns the total number of sockets provided by the specified adapter.

The parameter **NumWindows** returns the total number of windows provided by the specified adapter.

The parameter **NumEDCs** returns the total number of EDCs provided by the specified adapter.

The parameter **NumBridgeWindows** returns the total number of bridge windows provided by the specified adapter.

EXAMPLE

```
#include          “pcc_mng.h”

...

unsigned long     result;
char              pBuffer[50];
AISTRUCT          *pAIStruct;
COUNT             NumSockets,
                  NumWindows,
                  NumEDCs,
                  NumBridgeWindows;

...

/*-------------------------------------------------------
  Get capabilities of adapter 4
  -------------------------------------------------------*/
pAIStruct = (AISTRUCT*)pBuffer;
pAIStruct->wBufferLength = 50;       /* Setup Bufferlength */
result = pcc_entry (    PCC_INQ_ADAPTER,
                        (unsigned long)4,
                        (unsigned long)pBuffer,
                        (unsigned long)&NumSockets,
                        (unsigned long)&NumWindows,
                        (unsigned long)&NumEDCs,
                        (unsigned long)&NumBridgeWindows);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR	no error
PCC_BAD_ADAPTER	invalid adapter

4.1.3 GetAdapter

NAME

GetAdapter reads the actual configuration of the specified adapter.

FUNCTION CODE

PCC_GET_ADAPTER (0x85)

PARAMETERS

ADAPTER Adapter

FLAGS8 *State

IRQ *SCRouting

DESCRIPTION

The configuration of the specified adapter is read. The parameter values are compatible to the parameters of the *SetAdapter* function.

PARAMETERS

The parameter **Adapter** specifies the PC Card adapter.

The parameter **State** will be filled with flags specifying the actual configuration. The following flags are defined:

PCC_AS_POWERDOWN	Hardware is in power conserve mode.
PCC_AS_MAINTAIN	Adapter and socket configuration are maintained while in power conserve mode.

The parameter **SCRouting** will return the actual setup for socket interrupts. The following flags are defined:

PCC_IRQ_HIGH	The status change interrupt is active high.
PCC_IRQ_ENABLE	The status change interrupt is enabled.

EXAMPLE

```
#include          “pcc_mng.h”

...

unsigned long     result;
FLAGS8            State;
IRQ               SCRouting;

...

/*-------------------------------------------------------
  Get actual configuration of adapter 4
  -------------------------------------------------------*/
result = pcc_entry (    PCC_GET_ADAPTER,
                        (unsigned long)4,
                        (unsigned long)&State,
                        (unsigned long)&SCRouting);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR	no error
PCC_BAD_ADAPTER	invalid adapter

4.1.4 SetAdapter

NAME

SetAdapter configures the specified adapter.

FUNCTION CODE

PCC_SET_ADAPTER (0x86)

PARAMETERS

ADAPTER	Adapter
FLAGS8	State
IRQ	SCRouting

DESCRIPTION

The specified adapter will be configured with this function. This function should be called before any sockets, windows or pages are set.

PARAMETERS

The parameter **Adapter** specifies the PC Card adapter.

The parameter **State** specifies the new configuration of the adapter. The following flags are defined:

PCC_AS_POWERDOWN	Hardware is set to power conserve mode.
PCC_AS_MAINTAIN	Adapter and socket configuration are maintained while in power conserve mode.

The parameter **SCRouting** specifies how socket interrupts will be setup. The following flags are defined:

PCC_IRQ_HIGH	The status change interrupt is active high.
PCC_IRQ_ENABLE	The status change interrupt is enabled.

EXAMPLE

```
#include           “pcc_mng.h”

...

unsigned long      result;

...

/*-------------------------------------------------------
  Enable status change interrupt for adapter 4
  -------------------------------------------------------*/
result = pcc_entry (    PCC_SET_ADAPTER,
                        (unsigned long)4,
                        (unsigned long)0,
                        (unsigned long)PCC_IRQ_ENABLE);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR	no error
PCC_BAD_ADAPTER	invalid adapter
PCC_BAD_IRQ	unsupported state or IRQ level specified

4.1.5 InquireWindow

NAME

InquireWindow gets the capabilities of the specified window.

FUNCTION CODE

PCC_INQ_WINDOW (0x87)

PARAMETERS

ADAPTER	Adapter
WINDOW	Window
PTR	pBuffer
FLAGS8	*WndCaps
SKTBITS	*Sockets

DESCRIPTION

This function informs about the capabilities of the specified window and for which sockets the window can be used.

PARAMETERS

The parameter **Adapter** specifies the PC Card adapter.

The parameter **Window** specifies the window on the specified PC Card adapter.

The parameter **pbuffer** is a pointer to the *WISTRUCT* structure where the window capabilities and limits are returned.

Data structure *WISTRUCT*:

WORD	wBufferLength	This is an input entry and must be set to the maximal length of the WinTable array.
WORD	wDataLength	The function will return in this entry to the real length of the CharTable array.
WINTBL	WinTable[1]	*WINTBL* is defined as a union where different information are set for memory in a *MEMWINTBL* structure or for I/O windows in a *IOWINTBL*. If the use as memory and I/O window is possible for the window, the memory description is always the first in the array.

Data structure *MEMWINTBL*:

FLAGS16	MemWndCaps	Flags specifying special abilities of the specified memory window. Defined flags are:	
		PCC_WC_BASE	Programmable base address for the window
		PCC_WC_SIZE	Programmable size for the window
		PCC_WC_WENABLE	Window can be enabled or disabled without reprogramming the characteristics.
		PCC_WC_8BIT	Window can be programmed for 8 bit data transfers.
		PCC_WC_16BIT	Window can be programmed for 16 bit data transfers
		PCC_WC_BALIGN	The base address must be aligned to a multiple of the window size.
		PCC_WC_POW2	The window size must be a value power of two.
		PCC_WC_CALIGN	The offsets must align to the size of the window.
		PCC_WC_PAVAIL	Paging is available on the hardware.
		PCC_WC_PSHARED	Paging hardware is shared with another window.
		PCC_WC_PENABLE	Paging can be enabled and disabled.
		PCC_WC_WP	The window can write protected.
BASE	FirstByte	Value defines the first accessible memory address of the specified window.	
BASE	LastByte	Value defines the last accessible memory address of the specified window.	
SIZE	MinSize	Value defines the minimum size for the specified memory window.	
SIZE	MaxSize	Value defines the maximum size for the specified memory window.	
SIZE	RepGran	Memory sizes have to be set to a multiple of the size returned in this entry.	
SIZE	RepBase	Memory bases have to start at a multiple of this value.	
SIZE	RepOffset	Memory offsets have to be set to a multiple of this value.	

SPEED	Slowest	Entry shows the slowest possible memory access. The value is specified in the data structure *SPEED* which is described below.
SPEED	Fastest	Entry shows the fastest possible memory access. The value is specified in the data structure *SPEED* which is described below.

Data structure *SPEED*:

BYTE	speed	This field specifies the speed or indicates that the extended speed setting shall be used. Possible values are defined for this field:

Code	Meaning
0	reserved
1	250 nsec
2	200 nsec
3	150 nsec
4	100 nsec
5..6	reserved
7	use extended speed

BYTE	extSpeedMant	This field specifies the mantissa of the extended speed setting. How this value is coded, is described below.
BYTE	extSpeedExp	This field specifies the exponent of the extended speed setting. The values are defined as follows:

Code	extSpeedMant	extSpeedMant
0x0	reserved	1 ns
0x1	1.0	10 ns
0x2	1.2	100 ns
0x3	1.3	1 µs
0x4	1.5	10 µs
0x5	2.0	100 µs
0x6	2.5	1 ms
0x7	3.0	10 ms
0x8	3.5	
0x9	4.0	
0xA	4.5	
0xB	5.0	
0xC	5.5	
0xD	6.0	
0xE	7.0	
0xF	8.0	

Data structure *IOWINTBL*:

FLAGS16	IOWndCaps	Flags specifying special abilities of the specified I/O window. Defined flags are: PCC_WC_BASE: Programmable base address for the window PCC_WC_SIZE: Programmable size for the window PCC_WC_WENABLE: Window can be enabled or disabled without reprogramming the characteristics. PCC_WC_8BIT: Window can be programmed for 8 bit data transfers. PCC_WC_16BIT: Window can be programmed for 16 bit data transfers. PCC_WC_BALIGN: Base address must be aligned to a multiple of the window size. PCC_WC_POW2: Window size must be a value power of two. PCC_WC_INPACK: Window allows the INPACK# signal. PCC_WC_EISA: Window allows EISA-like I/O mapping. PCC_WC_CENABLE: EISA-like common address space may be ignored.
BASE	FirstByte	Value defines the first accessible I/O address of the specified window.
BASE	LastByte	Value defines the last accessible I/O address of the specified window.
SIZE	MinSize	Value defines the minimum size for the specified I/O window.
SIZE	MaxSize	Value defines the maximum size for the specified I/O window.
SIZE	RepGran	I/O sizes have to be set to a multiple of the size returned in this entry.
COUNT	AddrLines	Entry returns the number of interpreted address lines at the controller, typically ten (10) or sixteen (16).
FLAGS8	EISASlot	Entry is used for EISA bus settings, so it will be unused on TPMC871 modules

The parameter **WndCaps** will be filled with flags describing the capabilities of the window. The following flags are defined for this value:

PCC_WC_COMMON	The window can map PC Card common memory into system memory space.
PCC_WC_ATTRIBUTE	The window can map PC Card attribute memory into system memory space.
PCC_WC_IO	The window can map PC Card I/O ports into system I/O space.
PCC_WC_WAIT	The window supports the WAIT# signal.

The **Sockets** parameter is filled with bit field marking the sockets which can be accessed by the window.

EXAMPLE

```
#include          “pcc_mng.h”

...

unsigned long     result;
char              pBuffer[80];
WISTRUCT          *pWIStruct;
FLAGS8            WndCaps;
SKTBITS           Sockets;

...

/*-------------------------------------------------------
  Get capabilities of window 2 on adapter 4
  -------------------------------------------------------*/
pWIStruct = (WISTRUCT*) pBuffer;
pWIStruct->wBufferLength = 80;      /* Setup Bufferlength */
result = pcc_entry (    PCC_INQ_WINDOW,
                        (unsigned long)4,
                        (unsigned long)2,
                        (unsigned long)pBuffer,
                        (unsigned long)&WndCaps,
                        (unsigned long)&Sockets);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR	no error
PCC_BAD_ADAPTER	invalid adapter
PCC_BAD_WINDOW	invalid window

4.1.6 GetWindow

NAME

GetWindow reads the actual configuration of the specified window.

FUNCTION CODE

PCC_GET_WINDOW (0x88)

PARAMETERS

ADAPTER	Adapter
WINDOW	Window
SOCKET	*Socket
SIZE	*Size
FLAGS8	*State
SPEED	*Speed
BASE	*Base

DESCRIPTION

The configuration of the specified window is read. The parameter values are compatible to the parameters of the *SetWindow* function.

PARAMETERS

The parameter **Adapter** specifies the PC Card adapter.

The parameter **Window** specifies the window of the specified PC Card adapter.

The parameter **Socket** returns the socket number the window is currently assigned to.

The **Size** parameter returns the current window size.

The **State** flag field returns the actual window state. The value can be a combination of the following defined flags:

PCC_WS_IO	The window maps registers from the PC Card into the host I/O space.
PCC_WS_ENABLED	The window is enabled for mapping card's address space into host memory or I/O space.
PCC_WS_16BIT	The window is programmed for 16 bit data width.
PCC_WS_PAGED	The window is divided into pages.
PCC_WS_EISA	The window is set for EISA I/O mapping.
PCC_WS_CENABLE	Accesses to I/O ports generates card enables.

The parameter **Speed** is a pointer to the data structure which will be filled with data specifying the current access speed of the window.

Data structure *SPEED*:

BYTE	speed	This field specifies the speed or indicates that the extended speed setting shall be used. Possible values are defined for this field:

Code	Meaning
0	reserved
1	250 nsec
2	200 nsec
3	150 nsec
4	100 nsec
5..6	reserved
7	use extended speed

BYTE	extSpeedMant	This field specifies the mantissa of the extended speed setting. How this value is coded, is described below.
BYTE	extSpeedExp	This field specifies the exponent of the extended speed setting. The values are defined as follows:

Code	extSpeedMant	extSpeedMant
0x0	reserved	1 ns
0x1	1.0	10 ns
0x2	1.2	100 ns
0x3	1.3	1 µs
0x4	1.5	10 µs
0x5	2.0	100 µs
0x6	2.5	1 ms
0x7	3.0	10 ms
0x8	3.5	
0x9	4.0	
0xA	4.5	
0xB	5.0	
0xC	5.5	
0xD	6.0	
0xE	7.0	
0xF	8.0	

The parameter **Base** specifies the base address the window is mapped to.

EXAMPLE

```
#include          “pcc_mng.h”

...

unsigned long     result;
FLAGS8            State;
SOCKET            Socket;
SPEED             Speed;
SIZE              Size;
BASE              Base;

...

/*-------------------------------------------------------
  Get actual configuration window 2 on adapter 4
  -------------------------------------------------------*/
result = pcc_entry (    PCC_GET_WINDOW,
                        (unsigned long)4,
                        (unsigned long)2,
                        (unsigned long)&Socket,
                        (unsigned long)&Size,
                        (unsigned long)&State,
                        (unsigned long)&Speed,
                        (unsigned long)&Base);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR	no error
PCC_BAD_ADAPTER	invalid adapter
PCC_BAD_WINDOW	invalid window

4.1.7 SetWindow

NAME

SetWindow configures the specified window.

FUNCTION CODE

PCC_SET_WINDOW (0x89)

PARAMETERS

ADAPTER	Adapter
WINDOW	Window
SOCKET	Socket
SIZE	Size
FLAGS8	State
SPEED	Speed
BASE	Base

DESCRIPTION

The specified window will be configured. This function allows the selection of a special memory area where the PC Card shall be mapped to.

PARAMETERS

The parameter **Adapter** specifies the PC Card adapter.

The parameter **Window** specifies the window of the specified PC Card adapter.

The parameter **Socket** specifies the socket the window is assigned to.

The **Size** parameter specifies the new size of the window.

The **State** flag field specifies the window configuration. The value can be a combination of the following defined flags, if they are valid for the window (see chapter “InquireWindow”):

PCC_WS_IO	The window maps registers from the PC Card into the host I/O space.
PCC_WS_ENABLED	The window shall be enabled for mapping card’s address space into host memory or I/O space. Memory access must also be enabled with the *SetPage* function (see chapter “SetPage”).
PCC_WS_16BIT	The window is programmed for 16 bit data width.
PCC_WS_PAGED	The window is divided into pages.
PCC_WS_EISA	The window is set for EISA I/O mapping.
PCC_WS_CENABLE	Accesses to I/O ports generates card enables.

The parameter **Speed** is a pointer to the data structure which will setup the access speed of the window. The TPMC871 allows access times up to 600nsec.

Data structure *SPEED*:

BYTE	speed	This field specifies the speed or indicates that the extended speed setting shall be used. Possible values are defined for this field:

Code	Meaning
0	reserved
1	250 nsec
2	200 nsec
3	150 nsec
4	100 nsec
5..6	reserved
7	use extended speed

BYTE	extSpeedMant	This field specifies the mantissa of the extended speed setting. How this value is coded, is described below.
BYTE	extSpeedExp	This field specifies the exponent of the extended speed setting. The values are defined as follows:

Code	extSpeedMant	extSpeedExp
0x0	reserved	1 ns
0x1	1.0	10 ns
0x2	1.2	100 ns
0x3	1.3	1 µs
0x4	1.5	10 µs
0x5	2.0	100 µs
0x6	2.5	1 ms
0x7	3.0	10 ms
0x8	3.5	
0x9	4.0	
0xA	4.5	
0xB	5.0	
0xC	5.5	
0xD	6.0	
0xE	7.0	
0xF	8.0	

The parameter **Base** specifies the base address the window will be mapped to.

EXAMPLE

```
#include          “pcc_mng.h”

...

unsigned long     result;
SPEED             Speed;

...

/*-------------------------------------------------------
  Set window 2 on adapter 4 at socket 0 with the following
  capabilities:
    - Size:  0x1000
    - State: enable window
    - Speed: 200ns
    - Base:  0xfd100000
  -------------------------------------------------------*/
Speed.speed        = 0x02;
Speed.extSpeedMant = 0;
Speed.extSpeedExp  = 0;
result = pcc_entry (    PCC_SET_WINDOW,
                        (unsigned long)4,
                        (unsigned long)2,
                        (unsigned long)0,
                        (unsigned long)0x1000,
                        (unsigned long)PCC_WS_ENABLED,
                        (unsigned long)&Speed,
                        (unsigned long)0xfd100000);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR	No error
PCC_BAD_ADAPTER	Invalid adapter
PCC_BAD_ATTRIBUTE	The requested state does not match to the windows capabilities.
PCC_BAD_BASE	The base address is invalid.
PCC_BAD_SIZE	The window size is invalid.
PCC_BAD_SOCKET	The window can not be assigned to the specified socket.
PCC_BAD_SPEED	Illegal speed specified
PCC_BAD_TYPE	*PCC_WS_IO* setting is invalid.
PCC_BAD_WINDOW	Invalid window

4.1.8 GetPage

NAME

GetPage reads the actual configuration of the specified page.

FUNCTION CODE

PCC_GET_PAGE (0x8A)

PARAMETERS

ADAPTER	Adapter
WINDOW	Window
SOCKET	Page
FLAGS8	*State
OFFSET	*Offset

DESCRIPTION

This function reads the configuration of the specified page for the specified window. The parameter values are compatible to the parameter values which have to be set in the *SetPage* function.

PARAMETERS

The parameter **Adapter** specifies the PC Card adapter.

The parameter **Window** specifies the window of the specified PC Card adapter.

The parameter **Page** specifies the page of the specified window.

The **State** flag field returns the actual page state. The value can be a combination of the following defined flags:

PCC_PS_ATTRIBUTE	The page maps attribute memory into the host systems common memory.
PCC_PS_ENABLED	The page is enabled. Access to the specified address area is only enabled if the window and page are enabled.
PCC_PS_WP	The paged area is write protected.

The parameter **Offset** specifies the PC Card local offset. This value specifies the local address on the PC Card.

EXAMPLE

```
#include      “pcc_mng.h”

...

unsigned long      result;
FLAGS8             State;
OFFSET             Offset;

...

/*-------------------------------------------------------
  Get actual configuration of page 0 for window 2 on
  adapter 4
  -------------------------------------------------------*/
result = pcc_entry (    PCC_GET_PAGE,
                        (unsigned long)4,
                        (unsigned long)2,
                        (unsigned long)0,
                        (unsigned long)&State,
                        (unsigned long)&Offset);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR	no error
PCC_BAD_ADAPTER	invalid adapter
PCC_BAD_PAGE	invalid page
PCC_BAD_WINDOW	invalid window

4.1.9 Set Page

NAME

SetPage configures the specified page on the specified window.

FUNCTION CODE

PCC_SET_PAGE (0x8B)

PARAMETERS

ADAPTER	Adapter
WINDOW	Window
SOCKET	Page
FLAGS8	State
OFFSET	Offset

DESCRIPTION

This function configures the page(s) of a window, selects the start address of the mapped area of the PC Card and enables or disables the PC Card accesses.

PARAMETERS

The parameter **Adapter** specifies the PC Card adapter.

The parameter **Window** specifies the window of the specified PC Card adapter.

The parameter **Page** specifies the page of the specified window.

The **State** flag field specifies the new page state. The value can be set as a combination of the following defined flags:

PCC_PS_ATTRIBUTE	The page shall map attribute memory into the host systems common memory.
PCC_PS_ENABLED	The page shall be enabled. Access to the specified address area is only enabled if the window and page are enabled.
PCC_PS_WP	The paged area shall be write protected.

The parameter **Offset** specifies the PC Card local offset. This value specifies the local address on the PC Card.

EXAMPLE

```
#include      “pcc_mng.h”

...

unsigned long      result;

...

/*-------------------------------------------------------
  Set actual configuration of page 0 for window 2 on
  adapter 4
    - Attribute mapping
    - page enable
    - offset = 0x100
  -------------------------------------------------------*/
result = pcc_entry (    PCC_SET_PAGE,
                        (unsigned long)4,
                        (unsigned long)2,
                        (unsigned long)0,
                        (unsigned long)
                             (PCC_PS_ATTRIBUTE |
                              PCC_PS_ENABLED),
                        (unsigned long)0x100);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR	no error
PCC_BAD_ADAPTER	invalid adapter
PCC_BAD_ATTRIBUTE	state with invalid attribute
PCC_BAD_OFFSET	offset is invalid
PCC_BAD_PAGE	invalid page
PCC_BAD_WINDOW	invalid window

4.1.10 InquireSocket

NAME

InquireSocket gets the capabilities of the specified socket.

FUNCTION CODE

PCC_INQ_SOCKET (0x8C)

PARAMETERS

ADAPTER	Adapter
SOCKET	Socket
PTR	pBuffer
FLAGS8	*SCIntCaps
FLAGS8	*SCRptCaps
FLAGS8	*CtlIndCaps

DESCRIPTION

This function describes the capabilities of the specified socket. Information about special abilities, possible interrupt routings and about the events creating interrupts or status changes will be returned.

PARAMETERS

The parameter **Adapter** specifies the PC Card adapter.

The parameter **Socket** specifies the PC Card socket on the specified adapter.

The parameter **pBuffer** is a pointer to the *SISTRUCT* structure where the socket capabilities will be stored.

Data structure *SISTRUCT*:

WORD	wBufferLength	This is an input entry and must be set to the maximal length of the *CharTable*.
WORD	wDataLength	The function will return in this entry to the real length of the *CharTable*.
SCHARTBL	CharTable	This structure holds the information about the interrupt and special abilities of the specified PC Card socket.

Data structure *SCHARTBL*:

FLAGS16	SktCaps	Flags specifying special capabilities of the specified I/O socket. Defined Flags are:	
		PCC_IF_MEMORY	Socket supports memory-only interface.
		PCC_IF_IO	Socket supports I/O and memory interface.
		PCC_IF_CB	Socket supports CardBus PC Card.
		PCC_IF_33VCC	Socket supports 3.3 V interface.
		PCC_IF_XXVCC	Socket supports X.X V interface.
		PCC_IF_VSKAY	Socket supports Low Voltage Key.
		PCC_IF_DMA	Socket supports 16 bit PC Card DMA transfers.

The parameter **SCIntCaps** returns which events can trigger the status change interrupt and the parameter **SCRptCaps** returns which events can be reported. A combination of the following defined flags will be returned:

PCC_SBM_WP	PC Card write protect
PCC_SBM_LOCKED	External generated indicating the state of an electrical or mechanical lock mechanism.
PCC_SBM_EJECT	External generated indicating a request to eject the card.
PCC_SBM_INSERT	External generated indicating a request to insert the card.
PCC_SBM_BVD1	Battery voltage detect1, indicator for battery is unserviceable
PCC_SBM_BVD2	Battery voltage detect2, indicator for battery is weak
PCC_SBM_RDYBSY	Indicator for Ready/Busy
PCC_SBM_CD	Card detect 1 and 2

The parameter **CtlIndCaps** returns which controls and indicators are supported for the socket. A combination of the following defined flags will be returned:

PCC_SBM_WP	PC Card write protect state
PCC_SBM_LOCKED	External generated indicator for the state of an externally mechanical or electrical for the state of an electrical or mechanical lock mechanism
PCC_SBM_EJECT	Control for motor to eject a PC Card
PCC_SBM_INSERT	Control for motor to insert a PC Card
PCC_SBM_LOCK	Control for Card Lock
PCC_SBM_BATT	Indication for BVD1 and BVD2
PCC_SBM_BUSY	Indicator for showing the card in-use
PCC_SBM_XIP	Indicator for eXecution-In-Place application in progress

EXAMPLE

```
#include          “pcc_mng.h”

...

unsigned long     result;
char              pBuffer[50];
SISTRUCT          *pSIStruct;
FLAGS8            SCIntCaps,
                  SCRptCaps,
                  CtlIndCaps;

...

/*-------------------------------------------------------
  Get capabilities of Socket 0 on adapter 4
  -------------------------------------------------------*/
pSIStruct = (SISTRUCT*)pBuffer;
pSIStruct->wBufferLength = 50;      /* Setup Bufferlength */
result = pcc_entry (    PCC_INQ_SOCKET,
                        (unsigned long)4,
                        (unsigned long)0,
                        (unsigned long)pBuffer,
                        (unsigned long)&SCIntCaps,
                        (unsigned long)&SCRptCaps,
                        (unsigned long)&CtlIndCaps);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR	no error
PCC_BAD_ADAPTER	invalid adapter
PCC_BAD_SOCKET	invalid socket

4.1.11 GetSocket

NAME

GetSocket reads the actual configuration of the specified socket.

FUNCTION CODE

PCC_GET_SOCKET (0x8D)

PARAMETERS

ADAPTER	Adapter
SOCKET	Socket
FLAGS8	*SCIntMask
PWRINDEX	*Vcontrol
PWRINDEX	*VccLevel
PWRINDEX	*VppLevels
FLAGS8	*State
FLAGS8	*CtlInd
IRQ	*IREQRouting
FLAGS8	*IFType
WORD	*IFIndex

DESCRIPTION

This function reads the actual configuration of the specified socket. The parameters are compatible to the parameters which have to be set in the *SetSocket function*.

PARAMETERS

The parameter **Adapter** specifies the PC Card adapter.

The parameter **Socket** specifies the socket on the specified PC Card adapter.

The parameter **SCIntMask** returns the events generated by status changes when they occur on the socket. The value can be a combination of the following values:

PCC_SBM_WP	PC Card write protect
PCC_SBM_LOCKED	External generated indicating the state of an electrical or mechanical lock mechanism
PCC_SBM_EJECT	External generated indicating a request to eject the card
PCC_SBM_INSERT	External generated indicating a request to insert the card
PCC_SBM_BVD1	Battery voltage detect1, indicator for battery is unserviceable
PCC_SBM_BVD2	Battery voltage detect2, indicator for battery is weak
PCC_SBM_RDYBSY	Indicator for Ready/Busy
PCC_SBM_CD	Card detect 1 and 2

The parameter **Vcontrol** returns voltage control settings. This can be a combination of the following defined values:

PCC_VXTL_CISREAD	Vcc and Vpp are controlled by the Vcc and Vpp fields.
PCC_VCTL_OVERRIDE	If set the Vcc level does not match to the value indicated by the voltage sense

(The following values are mutually exclusive)

PCC_VCTL_50V	Use 5.0V for CIS read
PCC_VCTL_33V	Use 3.3V for CIS read
PCCVCTL_XXV	Use X.XV for CIS read

The parameter **VccLevel** specifies an index into the *PWRENTRY* array returned by the *InquireAdapter* function (see chapter “InquireAdapter”). The used index will specify the voltage which is used for Vcc.

The parameter **VppLevels** is an array of indices into the *PWRENTRY* array returned by the *InquireAdapter* function (see chapter “InquireAdapter”). The used indices will specify the voltages which are used for Vpp1 and Vpp2.

The parameter **State** returns the latched value of the state changes experienced on the specified socket. The values must be explicitly cleared by using the *SetSocket* function (see chapter “SetSocket”). The parameter is a combination of the following defined values:

PCC_SBM_WP	PC Card is write protected.
PCC_SBM_LOCKED	External generated indicating the state of an electrical or mechanical lock mechanism
PCC_SBM_EJECT	External generated indicating a request to eject the card
PCC_SBM_INSERT	External generated indicating a request to insert the card
PCC_SBM_BVD1	Battery voltage detect1, indicator for battery is unserviceable
PCC_SBM_BVD2	Battery voltage detect2, indicator for battery is weak
PCC_SBM_RDYBSY	Indicator for Ready/Busy
PCC_SBM_CD	Card detect 1 and 2

The parameter **CtlInd** returns the current setting of the socket controls and indicators. The following values are defined for this parameter:

PCC_SBM_WP	PC Card write protect state
PCC_SBM_LOCKED	External generated indicator for the state of an externally mechanical or electrical for the state of an electrical or mechanical lock mechanism
PCC_SBM_EJECT	Control for motor to eject a PC Card
PCC_SBM_INSERT	Control for motor to insert a PC Card
PCC_SBM_LOCK	Control for Card Lock
PCC_SBM_BATT	Indication for BVD1 and BVD2
PCC_SBM_BUSY	Indicator for showing the card in-use
PCC_SBM_XIP	Indicator for eXecution-In-Place application in progress

The parameter **IREQRouting** returns the actual IREQ signal. The can be a combination of the following values:

PCC_IRQ_HIGH	The PC Card interrupt is active high.
PCC_IRQ_ENABLE	The PC Card interrupt is enabled.

The parameter **IFType** returns the current interface setting. The following values are defined.

PCC_IF_MEMORY	Socket is set to memory-only interface.
PCC_IF_IO	Socket is set to I/O and memory interface.
PCC_IF_CARDBUS	Socket is set to CardBus PC Card.
PCC_IF_CUSTOM	The socket will use the custom interface described with the parameter **IFIndex**..
PCC_DREQ	A binary value will describe which DMA channel is currently used for DREQ#.

The parameter **IFIndex** returns the index into the custom interface array.

EXAMPLE

```
#include      “pcc_mng.h”

...

unsigned long      result;
FLAGS8             State,
                   SCIntMask,
                   CtlInd,
                   IFType;
IRQ                IREQRouting;
PWRINDEX           Vcontrol, VccLevel, VppLevels[2];
WORD               IFIndex;

...

/*-------------------------------------------------------
  Get actual configuration of socket 0 on adapter 4
  -------------------------------------------------------*/
result = pcc_entry (    PCC_GET_SOCKET,
                        (unsigned long)4,
                        (unsigned long)0,
                        (unsigned long)&SCIntMask,
                        (unsigned long)&Vcontrol,
                        (unsigned long)&VccLevel,
                        (unsigned long)VppLevels,
                        (unsigned long)&State,
                        (unsigned long)&CtlInd,
                        (unsigned long)&IREQRouting,
                        (unsigned long)&IFType,
                        (unsigned long)&IFIndex);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}
...
```

RETURNS

PCC_NO_ERROR	no error
PCC_BAD_ADAPTER	invalid adapter
PCC_BAD_SOCKET	invalid socket

4.1.12 SetSocket

NAME

SetSocket sets a new configuration for the specified socket.

FUNCTION CODE

PCC_SET_SOCKET (0x8E)

PARAMETERS

ADAPTER	Adapter
SOCKET	Socket
FLAGS8	SCIntMask
PWRINDEX	Vcontrol
PWRINDEX	VccLevel
PWRINDEX	*VppLevels
FLAGS8	State
FLAGS8	CtlInd
IRQ	IREQRouting
FLAGS8	IFType
WORD	IFIndex

DESCRIPTION

This function reads the actual configuration of the specified socket. The parameters are compatible to the parameters which have to be set in the *SetSocket function*.

PARAMETERS

The parameter **Adapter** specifies the PC Card adapter.

The parameter **Socket** specifies the socket on the specified PC Card adapter.

The parameter **SCIntMask** sets the events which shall generate a status change interrupt when they occur on the socket. The value can be a combination of the following values:

PCC_SBM_WP	PC Card write protect
PCC_SBM_LOCKED	External generated indicating the state of an electrical or mechanical lock mechanism
PCC_SBM_EJECT	External generated indicating a request to eject the card
PCC_SBM_INSERT	External generated indicating a request to insert the card
PCC_SBM_BVD1	Battery voltage detect1, indicator for battery is unserviceable
PCC_SBM_BVD2	Battery voltage detect2, indicator for battery is weak
PCC_SBM_RDYBSY	Indicator for Ready/Busy
PCC_SBM_CD	Card detect 1 and 2

The parameter **Vcontrol** specifies how to handle the voltage control. This can be a combination of the following defined values:

PCC_VXTL_CISREAD	Vcc and Vpp are controlled by the Vcc and Vpp fields.
PCC_VCTL_OVERRIDE	If set the Vcc level does not match to the value indicated by the voltage sense

The parameter **VccLevel** specifies an index into the *PWRENTRY* array returned by the *InquireAdapter* function (see chapter4.1.2 “InquireAdapter”). The used index will specify the voltage which will be set used for Vcc.

The parameter **VppLevels** is an array of indices into the *PWRENTRY* array returned by the *InquireAdapter* function (see chapter “InquireAdapter”). The used indices will specify the voltages which will be set for Vpp1 and Vpp2.

The parameter **State** clears the specified flags in the latched value of the state changes experienced on the specified socket. The parameter is a combination of the following defined values:

PCC_SBM_WP	PC Card is write protected.
PCC_SBM_LOCKED	External generated indicating the state of an electrical or mechanical lock mechanism
PCC_SBM_EJECT	External generated indicating a request to eject the card
PCC_SBM_INSERT	External generated indicating a request to insert the card
PCC_SBM_BVD1	Battery voltage detect1, indicator for battery is unserviceable
PCC_SBM_BVD2	Battery voltage detect2, indicator for battery is weak
PCC_SBM_RDYBSY	Indicator for Ready/Busy
PCC_SBM_CD	Card detect 1 and 2

The parameter **CtlInd** sets up the setting of the socket controls and indicators. The following values are defined for this parameter:

PCC_SBM_WP	PC Card write protect state
PCC_SBM_LOCKED	External generated indicator for the state of an externally mechanical or electrical the state of an electrical or mechanical lock mechanism
PCC_SBM_EJECT	Control for motor to eject a PC Card
PCC_SBM_INSERT	Control for motor to insert a PC Card
PCC_SBM_LOCK	Control for Card Lock
PCC_SBM_BATT	Indication for BVD1 and BVD2
PCC_SBM_BUSY	Indicator for showing the card in-use
PCC_SBM_XIP	Indicator for eXecution-In-Place application in progress

The parameter **IREQRouting** sets the IREQ# routing. The value can be a combination of the following values:

PCC_IRQ_HIGH	The PC Card interrupt is active high.
PCC_IRQ_ENABLE	The PC Card interrupt is enabled.

The parameter **IFType** specifies the interface setting. The following values are defined:

PCC_IF_MEMORY	Socket is set to memory-only interface.
PCC_IF_IO	Socket is set to I/O and memory interface.
PCC_IF_CARDBUS	Socket is set to CardBus PC Card.
PCC_IF_CUSTOM	The socket will use the custom interface described with the parameter **IFIndex**.
PCC_DREQ	A binary value will describe which DMA channel is currently used for DREQ#.

The parameter **IFIndex** specifies an index into the custom interface array. This value is only valid if *PCC_IF_CUSTOM* is specified.

EXAMPLE

```
#include      “pcc_mng.h”

...

unsigned long      result;
PWRINDEX           VppLevels[2];

...

/*-------------------------------------------------------
  Set a new configuration for socket 0 on adapter 4
    - enable status change interrupts for Card Detect
    - use PowerIndex 0 for Vcc and PowerIndex 1 for Vpp1/2
    - memory interface
  -------------------------------------------------------*/
VppLevels[0] = 0;
VppLevels[1] = 0;
result = pcc_entry (    PCC_SET_SOCKET,
                        (unsigned long)4,
                        (unsigned long)0,
                        (unsigned long)PCC_SBM_CD,
                        (unsigned long)0,
                        (unsigned long)2,
                        (unsigned long)VppLevels,
                        (unsigned long)0,
                        (unsigned long)0,
                        (unsigned long)0,
                        (unsigned long)PCC_IF_MEMORY,
                        (unsigned long)0);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR	no error
PCC_BAD_ADAPTER	invalid adapter
PCC_BAD_IRQ	specified IRQRouting is not defined
PCC_BAD_SOCKET	invalid socket
PCC_BAD_TYPE	invalid IFType
PCC_BAD_VCC	invalid Vcc level
PCC_BAD_VPP	invalid Vpp level defined
PCC_BAD_ATTRIBUTE	invalid *VCTL_x* combination

4.1.13 GetStatus

NAME

GetStatus reads status information of the specified socket

FUNCTION CODE

PCC_GET_STATUS (0x8F)

PARAMETERS

ADAPTER	Adapter
SOCKET	Socket
FLAGS8	*CardState
FLAGS8	*SocketState
FLAGS8	*CtlInd
IRQ	*IREQRouting
FLAGS8	*IFType

DESCRIPTION

This function reads the status of the card, socket, controls and indicators of the specified socket.

This service should not be invoked during hardware interrupt processing. It is intended to be used on application layer.

PARAMETERS

The parameter **Adapter** specifies the PC Card adapter.

The parameter **Socket** specifies the socket on the specified PC Card adapter.

The parameter **CardState** returns the actual state of the socket and the PC Card. The returned values are dependent from the capabilities of the PC Card controller. The value can be a combination of the following values:

PCC_SBM_WP	PC Card write protect
PCC_SBM_LOCKED	External generated indicating the state of an electrical or mechanical lock mechanism
PCC_SBM_EJECT	External generated indicating a request to eject the card
PCC_SBM_INSERT	External generated indicating a request to insert the card
PCC_SBM_BVD1	Battery voltage detect1, indicator for battery is unserviceable
PCC_SBM_BVD2	Battery voltage detect2, indicator for battery is weak
PCC_SBM_RDYBSY	Indicator for Ready/Busy
PCC_SBM_CD	Card detect 1 and 2

The parameter **SocketState** returns the latched value of the state changes experienced on the specified socket. The values must be explicitly cleared by using the *SetSocket* function (see chapter “SetSocket”). The parameter is a combination of the following defined values:

PCC_SBM_WP	PC Card is write protected.
PCC_SBM_LOCKED	External generated indicating the state of an electrical or mechanical lock mechanism
PCC_SBM_EJECT	External generated indicating a request to eject the card
PCC_SBM_INSERT	External generated indicating a request to insert the card
PCC_SBM_BVD1	Battery voltage detect1, indicator for battery is unserviceable
PCC_SBM_BVD2	Battery voltage detect2, indicator for battery is weak
PCC_SBM_RDYBSY	Indicator for Ready/Busy
PCC_SBM_CD	Card detect 1 and 2

The parameter **CtlInd** returns the current setting of the socket controls and indicators. The following values are defined for this parameter:

PCC_SBM_WP	PC Card write protect state
PCC_SBM_LOCKED	External generated indicator for the state of an externally mechanical or electrical the state of an electrical or mechanical lock mechanism
PCC_SBM_EJECT	Control for motor to eject a PC Card
PCC_SBM_INSERT	Control for motor to insert a PC Card
PCC_SBM_LOCK	Control for Card Lock
PCC_SBM_BATT	Indication for BVD1 and BVD2
PCC_SBM_BUSY	Indicator for showing the card in-use
PCC_SBM_XIP	Indicator for eXecution-In-Place application in progress

The parameter **IREQRouting** returns the actual IREQ signal. The value can be a combination of the following values:

PCC_IRQ_HIGH	The PC Card interrupt is active high.
PCC_IRQ_ENABLE	The PC Card interrupt is enabled.

The parameter **IFType** returns the current interface setting. The following values are defined.

PCC_IF_MEMORY	Socket is set to memory-only interface.
PCC_IF_IO	Socket is set to I/O and memory interface.
PCC_IF_CARDBUS	Socket is set to CardBus PC Card.
PCC_IF_CUSTOM	The socket will use the custom interface.
PCC_DREQ	A binary value will describe which DMA channel is currently used for DREQ#.

EXAMPLE

```
#include      “pcc_mng.h”
...
unsigned long      result;
FLAGS8             CardState,
                   SocketState,
                   CtlInd,
                   IFType;
IRQ                IREQRouting;

...
/*-------------------------------------------------------
  Get actual status of socket 0 on adapter 4
  -------------------------------------------------------*/
result = pcc_entry (    PCC_GET_STATUS,
                        (unsigned long)4,
                        (unsigned long)0,
                        (unsigned long)&CardState,
                        (unsigned long)&SocketState,
                        (unsigned long)&CtlInd,
                        (unsigned long)&IREQRouting,
                        (unsigned long)&IFType);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR	no error
PCC_BAD_ADAPTER	invalid adapter
PCC_BAD_SOCKET	invalid socket

4.1.14 ResetSocket

NAME

ResetSocket resets PC Card and socket hardware.

FUNCTION CODE

PCC_RESET_SOCKET (0x90)

PARAMETERS

ADAPTER	Adapter
SOCKET	Socket

DESCRIPTION

This function resets the PC Card in the specified socket and it resets the specified socket hardware to its power-on default state.

PARAMETERS

The parameter **Adapter** specifies the PC Card adapter.

The parameter **Socket** specifies the socket on the specified PC Card adapter.

EXAMPLE

```
#include      “pcc_mng.h”

...

unsigned long result;

...

/*-------------------------------------------------------
  Reset PC Card and hardware on socket 0 on adapter 4
  -------------------------------------------------------*/
result = pcc_entry (    PCC_RESET_SOCKET,
                        (unsigned long)4,
                        (unsigned long)0);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR	no error
PCC_BAD_ADAPTER	invalid adapter
PCC_BAD_SOCKET	invalid socket
PCC_NO_CARD	no card in specified socket

4.1.15 GetVendorInfo

NAME

GetVendorInfo gets information about the vendor implementing of the socket functions.

FUNCTION CODE

PCC_GET_VENDOR_INFO (0x9D)

PARAMETERS

ADAPTER	Adapter
BYTE	Type
PTR	*pBuffer
BCD	*Release

DESCRIPTION

This function returns information about the socket functions.

PARAMETERS

The parameter **Adapter** specifies the PC Card adapter.

The parameter **Type** specifies the type of the client-supplied buffer. The only currently defined **Type** is zero (0).

The parameter **pBuffer** is a pointer to the *VISTRUCT* structure where the adapter capabilities will be stored.

Data structure *VISTRUCT*:

WORD	wBufferLength	This is an input entry and must be set to the maximal length of the *szImplementor* array.
WORD	wDataLength	The function will return in this entry to the real length of the *szImplementor* array.
char	szImplementor	This field is filled with an ASCII string, giving information about the vendor.

The parameter **Release** returns the release number of actual used socket functions. This value is BCD coded, 0x0100 means 1.00.

EXAMPLE

```
#include      “pcc_mng.h”

...

unsigned long      result;
char               pBuffer[80];
VISTRUCT           *pVIStruct;
BCD                Release;

...

/*-------------------------------------------------------
  Get vendor information for adapter 4
  -------------------------------------------------------*/
pVIStruct = (VISTRUCT*)&(pBuffer[0]);
pVIStruct->wBufferLength = 70;
result = pcc_entry (    PCC_GET_VENDOR_INFO,
                        (unsigned long)4,
                        (unsigned long)0,
                        (unsigned long)pBuffer,
                        (unsigned long)&Release);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR	no error
PCC_BAD_ADAPTER	invalid adapter
PCC_BAD_SERVICE	invalid service type

4.1.16 AcknowlegdeInterrupt

NAME

AcknowledgeInterrupt gets information of interrupt source and remove interrupt.

FUNCTION CODE

PCC_ACK_INTERRUPT (0x9E)

PARAMETERS

ADAPTER	Adapter
SKTBITS	*Sockets

DESCRIPTION

This function returns information on which sockets an interrupt is pending and clears the interrupt source. This function should always be called in the status change interrupt function.

PARAMETERS

The parameter **Adapter** specifies the PC Card adapter.

The parameter **Sockets** returns a bit-field specifying the sockets where an interrupt has occurred. Bit 0 is set for Socket 0, bit 1 is set for Socket 1 and so on.

EXAMPLE

```
#include      “pcc_mng.h”

...

unsigned long      result;
SKTBITS            Sockets;
...

/*-------------------------------------------------------
  Acknowledge interrupts on adapter 4
  -------------------------------------------------------*/
result = pcc_entry (    PCC_ACK_INTERRUPT,
                        (unsigned long)4,
                        (unsigned long)&Sockets);
if (result)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

PCC_NO_ERROR	no error
PCC_BAD_ADAPTER	invalid adapter

4.2 Unsupported Functions

These functions will all return the error code *TP871_SS_UNIMPL_FUNCTION*.

4.2.1 GetSSInfo

NAME

GetSSInfo gets information about the socket services.

FUNCTION CODE

PCC_GET_SS_INFO (0x83)

PARAMETERS

ADAPTER	Adapter
BCD	*Complience
COUNT	*NumAdapters
ADAPETER	*FirstAdapter

4.2.2 InquireEDC

NAME

InquireEDC gets capabilities of the EDC generator.

FUNCTION CODE

PCC_INQ_EDC (0x95)

PARAMETERS

ADAPTER	Adapter
EDC	Edc
SKTBITS	*Sockets
FLAGS8	*Caps
FLAGS8	*Types

4.2.3 GetEDC

NAME

GetEDC gets actual configuration of the specified EDC.

FUNCTION CODE

PCC_GET_EDC (0x96)

PARAMETERS

ADAPTER	Adapter
EDC	Edc
SOCKET	*Socket
FLAGS8	*State
FLAGS8	*Type

4.2.4 SetEDC

NAME

SetEDC sets new configuration for the specified EDC.

FUNCTION CODE

PCC_SET_EDC (0x97)

PARAMETERS

ADAPTER	Adapter
EDC	Edc
SOCKET	Socket
FLAGS8	State
FLAGS8	Type

4.2.5 StartEDC

NAME

StartEDC starts the previously configured EDC.

FUNCTION CODE

PCC_START_EDC (0x98)

PARAMETERS

ADAPTER	Adapter
EDC	Edc

4.2.6 PauseEDC

NAME

PauseEDC pauses the specified EDC.

FUNCTION CODE

PCC_PAUSE_EDC (0x99)

PARAMETERS

ADAPTER	Adapter
EDC	Edc

4.2.7 ResumeEDC

NAME

ResumeEDC resumes the specified, paused and configured EDC.

FUNCTION CODE

PCC_RESUME_EDC (0x9A)

PARAMETERS

ADAPTER	Adapter
EDC	Edc

4.2.8 StopEDC

NAME

StopEDC stops the specified EDC.

FUNCTION CODE

PCC_STOP_EDC (0x9B)

PARAMETERS

ADAPTER	Adapter
EDC	Edc

4.2.9 ReadEDC

NAME

ReadEDC reads the EDC value from the specified EDC.

FUNCTION CODE

PCC_READ_EDC (0x9C)

PARAMETERS

ADAPTER	Adapter
EDC	Edc
DWORD	*Value

4.2.10 GetSetPriorHandler

NAME

GetSetPriorHandler replaces or obtains the entry point of a prior handler for the adapter.

FUNCTION CODE

PCC_PRIOR_HANDLER (0x9F)

PARAMETERS

ADAPTER	Adapter
FLAGS8	Mode
PTR	pHandler

4.2.11 GetSetSSAddr

NAME

GetSetSSAddr returns code and data area descriptions.

FUNCTION CODE

PCC_SS_ADR (0xA0)

PARAMETERS

ADAPTER	Adapter
BYTE	Mode
BYTE	Subfunc
COUNT	*NumAddData
PTR	pBuffer

4.2.12 GetAccessOffsets

NAME

GetAccessOffsets

FUNCTION CODE

PCC_ACCESS_OFFSETS (0xA1)

PARAMETERS

ADAPTER	Adapter
BYTE	Mode
COUNT	NumDesired
PTR	pBuffer
COUNT	*NumAvail

4.2.13 AccessConfigurationSpace

NAME

AccessConfigurationSpace provides an interface to read and write in CardBus configuration space.

FUNCTION CODE

PCC_READ_EDC (0xA2)

PARAMETERS

ADAPTER	Adapter
SOCKET	Socket
BYTE	Function
FLAGS8	Action
OFFSET	Location
FLAGS32	*Data

4.2.14 InquireBridgeWindow

NAME

InquireBridgeWindow reads capabilities about the specified bridge window.

FUNCTION CODE

PCC_INQ_BWINDOW (0xA3)

PARAMETERS

ADAPTER	Adapter
WINDOW	window
PTR	pBuffer
FLAGS8	*WndCaps
SKTBITS	*Sockets

4.2.15 GetBridgeWindow

NAME

GetBridgeWindow gets actual configuration of the specified bridge window.

FUNCTION CODE

PCC_GET_BWINDOW (0xA4)

PARAMETERS

ADAPTER	Adapter
WINDOW	window
SOCKET	*Socket
SIZE	*Size
FLAGS8	*State
BASE	*Base

4.2.16 SetBridgeWindow

NAME

SetBridgeWindow sets new configuration of the specified bridge window.

FUNCTION CODE

PCC_SET_BWINDOW (0xA5)

PARAMETERS

ADAPTER	Adapter
WINDOW	window
SOCKET	Socket
SIZE	Size
FLAGS8	State
BASE	Base

4.2.17 VendorSpecific

NAME

VendorSpecific Function is vendor specific.

FUNCTION CODE

PCC_READ_EDC (0xAE)

PARAMETERS

ADAPTER Adapter
...

5 ATA Disk Device Driver

This VxWorks ATA Disk device driver allows the operation of PC Card ATA disks with TPMC871 PC Card PMC conforming to the VxWorks block I/O system specification. The ATA Disk device driver is an addition to the TPMC871-SW-42 Device Interface Software and will not work without it. It will call the calling interface which is a unified interface for PC Card interfaces from TEWS TECHNOLOGIES, which allows to run this driver on different PC Card interfaces.

Before using the ATA Disk driver the calling interface and the PC Card devices must be initialized.

This ATA Disk device driver is a standard block device and the implemented file systems of VxWorks shall be possible to be handled by the driver. More information on block devices and the file systems can be found in the VxWorks Programmer's Guide.

5.1 I/O System Functions

This chapter describes the driver level interface to the I/O system. The purpose of these functions is to install the driver in the I/O system, add and initialize devices.

5.1.1 pccAtaDrv()

NAME

pccAtaDrv() initializes ATA Disk PC Card driver.

SYNOPSIS

```
STATUS pccAtaDrv
(
    void
)
```

DESCRIPTION

This function initializes the local values of ATA Disk PC Card driver. This function must be called before any other function of this ATA Disk driver is called.

EXAMPLE

```
#include      “ata_drv.h”

...

STATUS        result;

...

result = pccAtaDrv();

if (result == ERROR)
{
     /* Handle the occurred error */
}
else
{
     /* Execution successful */
}

...
```

RETURNS

OK or ERROR if an error occurred. On ERROR the driver will set the global error value to an appropriate error code. For a full description of the error codes see chapter “Error Codes”.

5.1.2 pccAtaDevCreate()

NAME

pccAtaDevCreate() creates a device descriptor for ATA Disk in the specified PC Card adapter.

SYNOPSIS

```
unsigned long     pccAtaDevCreate
(
        ADAPTER                 adNo,
        SOCKET                  sockNo,
        int                     intLevel,
        int                     intVector,
        unsigned long           locWinOffset
)
```

DESCRIPTION

This function creates an ATA Disk device on the specified PC Card socket that will be serviced by the ATA Disk driver. This function must be called before accesses are made to the ATA Disk. The function will return a device descriptor, which must be specified when creating the device for the file system (for example *dosFsDevInit()*).

PARAMETERS

The parameters **adNo** and **sockNo** specify the PC Card socket where the ATA Disk device shall be created on.

The parameters **intLevel** and **intVector** specify the interrupt level and the interrupt vector for the interrupts created for the specified socket. The values of these levels are hardware and BSP dependent. The values which are specified in the BSP documentation for #INTA of the PMC slot the module is mounted have to be specified.

The parameter **locWinOffset** specifies the address offset in the PCI memory space. Every ATA Disk device will map two spaces into the PCI memory space, one space with a size of 0x400 byte and one space with a size of 0x10 byte. If it is not possible to create windows of this size, the driver will take the next possible size. On the TPMC871 there will be two spaces with a size of 0x1000 byte.

EXAMPLE

```
#include      “ata_drv.h”

...

unsigned long      dev_handle;

...

/*-------------------------------------------------------
  Create an ATA Disk device on adapter 4, socket 0. The
  interrupt level and interrupt vector are 0x19. The
  window offset shall be 0x4000
  -------------------------------------------------------*/
dev_handle = pccAtaDevCreate (4, 0, 0x19, 0x19, 0x4000);
if (dev_handle)
{
     /* Execution successful */
}
else
{
     /* Handle the occurred error */
}

...
```

RETURNS

This function returns a device descriptor on success and on ERROR it will return 0. On ERROR the driver will set the global error value to an appropriate error code. For a full description of the error codes see chapter “Error Codes”.

5.2 I/O Interface Functions

This chapter describes the interface to the basic I/O system.

5.2.1 open()

NAME

open() opens a device or file.

SYNOPSIS

```
int open
(
        const char      *name,      /* name of the file to open  */
        int             flags,      /* select access mode (O_RDONLY, O_WRONLY ...)      */
        int             mode        /* mode of file to create    */
)
```

DESCRIPTION

Before I/O can be performed to a file, a file descriptor must be opened by invoking the basic I/O function *open().*

EXAMPLE

```
...

/*-------------------------------------------------------
  Open the file named "example.txt" at the drive
  named "PC1:" for a read access
  -------------------------------------------------------*/
fd = open ("PC1:\\example.txt", O_RDONLY, 0);

...
```

RETURNS

A file descriptor number or ERROR (if the file does not exist or no file descriptors are available)

SEE ALSO

ioLib, basic I/O routine - *open()*

5.2.2 read()

NAME

read() reads bytes from the specified file.

SYNOPSIS

```
int read
(
        int         fd,             /* descriptor of opened file  */
        char        *buffer,        /* pointer to a buffer to receive bytes      */
        size_t      maxbytes        /* max number of bytes to read       */
)
```

DESCRIPTION

This routine reads a number of bytes from the specified file and places them in **buffer**. The parameter **maxbytes** specifies the maximum number of bytes to read.

EXAMPLE

```
int  fd;
int  nbytes;
char buffer[80];

...

/*----------------------------------------------------------
  Read up to 80 bytes from the file connected with
  the file descriptor fd
  ----------------------------------------------------------*/
nbytes = read (fd, buffer, 80);

...
```

RETURNS

ERROR or number of bytes read

SEE ALSO

ioLib, basic I/O routine - *read()*

5.2.3 write()

NAME

write() writes bytes to the specified file on ATA Disk device.

SYNOPSIS

```
int write
(
    int       fd,          /* file descriptor on which to write   */
    char      *buffer,     /* buffer containing bytes to be written   */
    size_t    nbytes       /* number of bytes to be written   */
)
```

PARAMETER

This routine writes **nbytes** bytes from **buffer** to the specified file connected with the file descriptor **fd**.

EXAMPLE

```
int  fd;
int  nbytes;

...

/*-------------------------------------------------------
  Write 12 bytes to the file connected with the
  file descriptor fd
  -------------------------------------------------------*/
nbytes = write (fd, "Hello world!", 12);

...
```

RETURNS

ERROR or number of bytes written

SEE ALSO

ioLib, basic I/O routine - *write()*

5.2.4 ioctl()

NAME

ioctl() performs an I/O control function.

SYNOPSIS

```
int ioctl
(
    int     fd,         /* file descriptor     */
    int     function,   /* function code       */
    int     arg         /* optional function dependent argument    */
    )
```

DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls (read, write) will be performed by calling the *ioctl()* function with a specific function code and an optional function dependent argument.

The ATA Disk driver supports no special I/O operation.

RETURNS

OK or ERROR (if the device descriptor does not exist or the function code is unknown)

INCLUDE FILES

ioLib.h

SEE ALSO

ioLib, basic I/O routine - *ioctl(),*VxWorks Programmer's Guide: I/O System and Local File System

6 Application Hints

This chapter gives a quick overview, what has to do before using the driver calls. The software example will be used and described. The example is taken from a MVME2300 application. With other hardware configuration it may be possible, that some setup values must be adapted.

6.1 Start up PC Card Socket Functions

This part shows how to startup the socket functions to be used from the application layer.

Specify the maximum number of adapters that shall be used in the system:

```
#define   NUM_ADA_DEVS              2
```

Define the configuration in the configuration table:

```
typedef struct
{
      int               busNo;
      int               devNo;
      int               funcNo;
      int               intLvl;
      int               intVec;
      ADAPTER           adapter;
} DEV_TAB_STRUCT;
```

Setup the configuration in the configuration table. **busNo**, **devNo** and **funcNo** are specifying the location of the TPMC871 module and they should be described in the carrier manual. The values of **intLvl** and **intVec** specify interrupt level and vector for the specified slot. These values are also defined in the BSP. The value of **adapter** specifies the adapter number, which will be used in the application.

For TPMC871 devices **funcNo** shall always be 0. For TPMC872 devices **funcNo** specifies the slot, 0 will specify the standard slot and 1 will select the backside slot.

The TPMC872 uses INTA only, therefore the values for intLvl and intVec of both slots must be the same.

```
LOCAL DEV_TAB_STRUCT        dev_tab[NUM_ADA_DEVS] =
{
     {0, 16, 0, 0x19, 0x19, 0},
     {0, 17, 0, 0x1C, 0x1C, 1}
};

...
```

Initialize the socket functions of the TPMC871:

```
result = tpmc871_ss_init();
if (result)
{
     printf("ERROR (%08lX)\n", result);
     return;
}
printf("OK\n");
```

Initialize the PC Card manager:

```
printf("Initialize PC Card manager ... ");
result = pcc_init();
if (result)
{
     printf("ERROR (%08lX)\n", result);
     return;
}
printf("OK\n");
```

Then connect the TPMC871 adapters to the PC Card manager. The easiest way to do this is to make a loop for all TPMC871 modules:

```
for (adNo = 0; adNo < NUM_ADA_DEVS; adNo++)
{
```

Initialize the TPMC871 (adapter) that is placed at the position specified in the configuration table. If there is a module mounted, the function will return a local adapter number, which is used by the TPMC871 socket functions and the entry point to call the TPMC871 socket functions.

```
     result = tpmc871_init(dev_tab[adNo].busNo,
          dev_tab[adNo].devNo,
          dev_tab[adNo].funcNo,
          &locAdapter,
          &locEntry);
     if (result)
     {
```

The initialization failed, check the result:

```
          printf("ERROR (%08lX)\n", result);
     }
     else
     {
          printf("OK\n");
```

The initialization was successful and now it has to be connected to PC Card manager. Specify adapter number used for this adapter and the two parameters received from the tpmc871_init() function.

```
          result = pcc_adapter_init(dev_tab[adNo].adapter,
               locAdapter,
               locEntry);
          if (result)
          {
```

The initialization failed, check the result:

```
                printf("ERROR (%08lX)\n", result);
        }
        else
        {
                printf("OK\n");
```

The TPMC871 is now accessible via the PC Card interface, if the advised adapter number is specified.

If intending to use interrupts, it is now a good time to connect the interrupt function into the VxWorks system. The third parameter specifies the adapter. This depends on the parameter of the interrupt service function.

```
                if( intConnect(
                        INUM_TO_IVEC(dev_tab[adNo].intVec),
                        (VOIDFUNCPTR)pccInterrupt,
                        (int)dev_tab[adNo].adapter) == ERROR)
                {
```

Connecting the interrupt failed, check errno:

```
                }
```

The interrupt level can now be enabled.

```
                intEnable(dev_tab[adNo].intLvl);
        }
    }
}

...
```

Then the unified socket functions are ready to use via the PC Card manager.

6.2 Start up the PC Card ATA Driver

This part shows what to do before using the PC Card ATA Disk with a DOS file system. Many part of the code are the same as in the simple initialization, only the differences are documented. For the other parts have a look to chapter “Start up PC Card Socket Functions”.

Specify the maximum number of files that shall be maximal opened at the same time:

```
#define MAX_FILES           10
#define NUM_ADA_DEVS        2

typedef struct
{
     int            busNo;
     int            devNo;
     int            funcNo;
     int            intLvl;
     int            intVec;
     ADAPTER        adapter;
} DEV_TAB_STRUCT;

LOCAL DEV_TAB_STRUCT    dev_tab[NUM_ADA_DEVS] =
{
     {0, 16, 0, 0x19, 0x19, 0},
     {0, 17, 0, 0x1C, 0x1C, 1}
};
```

Space for disk device handles must be allocated and the names for the ATA Disks have to be defined. After that space for the volume descriptors has to be allocated.

```
LOCAL unsigned long     drv_handle[NUM_ADA_DEVS][1];
LOCAL char              drv_name[NUM_ADA_DEVS][1][20] =
{
     {"/PCC1:"},
     {"/PCC2:"}
};
LOCAL DOS_VOL_DESC *vol_desc[NUM_ADA_DEVS][1];

...
```

Initialize the drivers and the dosFileSystem.

```
     if (dosFsInit(MAX_FILES) != OK)
     {
```

Initialization of the DOS Filesystem failed, check errno:

```
        printf("ERROR (0x%08X)\n", errnoGet());
        return;
    }
    printf("OK\n");

    /* Initialize Socket Service Handlers (TPMC871) */
    printf("Initialize Socket Services for TPMC871 ... ");
    result = tpmc871_ss_init();
    ...

    /* Initialize PC Card manager */
    printf("Initialize PC-Card Manager ... ");
    result = pcc_init();
    ...

    printf("\nConnect Adapters:\n");
    for (adNo = 0; adNo < NUM_ADA_DEVS; adNo++)
    {
        result = tpmc871_init(  dev_tab[adNo].busNo,
            dev_tab[adNo].devNo,
            dev_tab[adNo].funcNo,
            &locAdapter,
            &locEntry);
        if (result)
        {
            printf("ERROR (%08lX)\n", result);
        }
        else
        {
            printf("OK\n");

            result = pcc_adapter_init(dev_tab[adNo].adapter,
                      locAdapter,
                      locEntry);
            if (result)
            {
                printf("ERROR (%08lX)\n", result);
            }
            else
            {
                printf("OK\n");
            }
        }
    }
```

Initialize the ATA Disk driver:

```
result = pccAtaDrv();
if (result)
{
```

The initialization failed, check the result:

```
    printf("ERROR (%08lX)\n", result);
    return;
}
printf("OK\n");
```

Try to initialize a disk device at every socket by doing a loop over all sockets. Specify an offset for the windows of the sockets that is mapped into the allowed PCI memory area.

```
winOffset = 0;
for (adNo = 0; adNo < NUM_ADA_DEVS; adNo++)
{
```

Get a device handle for the new disk device and initialize the socket.

```
    drv_handle[adNo][0] = pccAtaDevCreate (
        adNo,
        sockNo,
        dev_tab[adNo].intLvl,
        dev_tab[adNo].intVec,
        winOffset);
    if (drv_handle[adNo][0])
    {
```

The initialization failed, check the result.

```
        printf("OK -- Handle: %08lX\n",
            drv_handle[adNo][0]);
    }
    else
    {
        printf("ERROR (%08Xh)\n", errnoGet());
    }
```

Count offset for the next module, be sure the PCI memory area is unused.

```
    winOffset += 0x2000;
}
```

Try to create disk volumes on the initialized adapters by doing it in a loop for all the successfully initialized adapters.

```
for (adNo = 0; adNo < NUM_ADA_DEVS; adNo++)
{
    if (drv_handle[adNo][0])
    {
```

Then create the disk volume.

```
                vol_desc[adNo][0] = dosFsDevInit(drv_name[adNo][0],
                     (BLK_DEV*)drv_handle[adNo][0],
                     NULL);
                if(vol_desc[adNo][0] == 0)
                {
```

Creation failed check errno.

```
                     printf("ERROR (0x%08X)\n", errnoGet());
                }
                else
                {
```

Creation successfully completed, now the disk shall be accessible.

```
                }
        }
 }
```

6.3 Setup Access Memory Card

This chapter gives a simple example how to setup a socket to allow memory accesses to a PC Card. The addresses depend on the used BSP and the BSP dependent setup of the used socket functions.

Example:

Map a window to a memory card. Place the socket (0) on adapter (2), using window (1). The windows address shall be at 0xfd100000 with a size of 0x1000. The PC Card internal offset shall be 0x0. The access shall be write protected. Setup the slot with fix parameters. In complex systems it will be better to read out the capabilities or the actual setup, to avoid conflicts.

Start after the initialization of the drivers:

First set up the adapter (2). Don't use any interrupts or any special modes.

```
result = pcc_entry (    PCC_SET_ADAPTER,
                        (unsigned long)2,
                        (unsigned long)0,
                        (unsigned long)0);
if (result)
{
     /* error */
}
```

Then set up the socket (0). Don't use any interrupts or any special modes. Set Vcc to 5V (Powerindex 2), Vpp is not needed. Use the memory interface.

```
VppLevels[0] = 0;       /* 0V */
VppLevels[1] = 0;       /* 0V */
result = pcc_entry (    PCC_SET_SOCKET,
                        (unsigned long)2,
                        (unsigned long)0,
                        (unsigned long)0,
                        (unsigned long)0,
                        (unsigned long)2,               /* 5 V */
                        (unsigned long)VppLevels,
                        (unsigned long)0,
                        (unsigned long)0,
                        (unsigned long)0,
                        (unsigned long)PCC_IF_MEMORY,
                        (unsigned long)0);
if (result)
{
     /* error */
}
```

Now it's time to configure the window (1). The size shall be 0x1000, access speed shall be 200ns, enable window access. Now the window is open, but the card access has to be set up with the *setPage* function.

```
speed.speed = 0x2;
result = pcc_entry(     PCC_SET_WINDOW,
                        (unsigned long)2,
                        (unsigned long)1,
                        (unsigned long)0,
                        (unsigned long)0x1000,      /* size */
                        (unsigned long)PCC_WS_ENABLE,
                        (unsigned long)&speed,
                        (unsigned long)0xFD100000); /* Baseaddr */
if (result)
{
     /* error */
}
```

At present set up the card access. Use page (0), because the controller doesn't allow paging. Read from the beginning of the card, that means offset zero (0). Enable the access and make it write protected.

```
result = pcc_entry (    PCC_SET_PAGE,
                             (unsigned long)2,
                        (unsigned long)1,
                        (unsigned long)0,
                        (unsigned long)PCC_PS_ENABLED | PCC_PS_WP,
                        (unsigned long)0); /* card offset */
```

The memory card can now be accessed at the address space from 0xfd100000 up to 0xfd100fff.

7 Appendix

7.1 Error Codes

This chapter gives a short description of the error codes, which can be created by this software interface.

7.1.1 TPMC871 Error Codes

TP871_NO_ERROR	0x00000000	Execution OK
TP871_NO_DEVICE	0x87100000	No TPMC871 found at specified slot
TP871_SS_NOT_INIT	0x87100001	The socket functions are not initialized yet.
TP871_SS_MAX_ADAPTER	0x87100002	Maximum number of adapters is already initialized.
TP871_SS_CANT_ALLOC_MEM	0x87100003	The socket functions can not allocate memory.
TP871_SS_ILL_FUNCTION	0x87100004	Illegal socket function code specified.
TP871_SS_UNIMPL_FUNCTION	0x87100005	Unimplemented socket function called.

7.1.2 Socket Function Error Codes

PCC_NO_ERROR	0x00000000	Execution OK
PCC_MAX_ADAPTER	0xF8710002	Maximum number of adapters is already connected.
PCC_BAD_ADAPTER	0xF8710101	Bad adapter number specified
PCC_BAD_ATTRIBUTE	0xF8710102	Bad combination of attributes
PCC_BAD_BASE	0xF8710103	Bad base specified
PCC_BAD_IRQ	0xF8710106	Bad IRQ state specified
PCC_BAD_OFFSET	0xF8710107	Bad offset specified
PCC_BAD_PAGE	0xF8710108	Bad page number specified
PCC_BAD_SIZE	0xF871010A	Bad size specified
PCC_BAD_SOCKET	0xF871010B	Bad socket number specified
PCC_BAD_TYPE	0xF871010D	Bad window type specified
PCC_BAD_VCC	0xF871010E	Bad Vcc index specified
PCC_BAD_VPP	0xF871010F	Bad Vpp index specified
PCC_BAD_WINDOW	0xF8710111	Bad window number specified
PCC_BAD_SERVICE	0xF8710115	Bad socket function specified
PCC_BAD_SPEED	0xF8710117	Bad speed specified

7.1.3 ATA Disk Driver Error Codes

S_pccAta_NOCARD	0xA8710001	No disk in the specified socket
S_pccAta_NOWINDOW	0xA8710002	Can not allocate a matching window
S_pccAta_WINSIZE	0xA8710003	Specified window size can not be mapped
S_pccAta_ATADISK	0xA8710004	No ATA Disk found at specified socket
S_pccAta_PARTERR	0xA8710005	Partition error
S_pccAta_ACCTIMEOUT	0xA8710006	Timeout while disk access
S_pccAta_ACCERROR	0xA8710007	Error occurred while disk access
S_pccAta_WINOFFSET	0xA8710008	Illegal window offset specified