

TPMC821-SW-65

Windows Device Driver

INTERBUS Master G4

Version 2.0.x

User Manual

Issue 2.0.0

July 2015

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

TPMC821-SW-65

Windows Device Driver

INTERBUS Master G4

Supported Modules:
TPMC821

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2002-2015 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0.0	First Issue	November 25, 2002
1.0.1	Slips of the pen rectified	December 13, 2002
2.0.0	General Revision, Windows 7 and 64-bit Support, Description of Installation removed	July 3, 2015

Table of Contents

1	INTRODUCTION.....	4
2	TPMC821 DEVICE DRIVER PROGRAMMING	5
2.1	TPMC821 Files and I/O Functions	5
2.1.1	Opening a TPMC821 Device	5
2.1.2	Closing a TPMC821 Device	7
2.1.3	TPMC821 Device I/O Control Functions	8
2.1.3.1	IOCTL_TP821_READ	10
2.1.3.2	IOCTL_TP821_WRITE	13
2.1.3.3	IOCTL_TP821_BIT_CMD	17
2.1.3.4	IOCTL_TP821_MBX_CMD, IOCTL_TP821_MBX_CMD_NOWAIT	18
2.1.3.5	IOCTL_TP821_GET_DIAG	19
2.1.3.6	IOCTL_TP821_CONFIG	21
2.1.3.7	IOCTL_TP821_SET_HOST_FAIL	23
2.1.3.8	IOCTL_TP821_RESET_HOST_FAIL	24
2.1.3.9	IOCTL_TP821_RESET_HARDWARE_FAIL	25
2.1.3.10	IOCTL_TP821_MOD_INFO	26

1 Introduction

The TPMC821-SW-65 Windows device driver is a kernel mode driver which allows the operation of the supported hardware module on an Intel or Intel-compatible Windows operating system.

The TPMC821-SW-65 device driver supports the following features:

- All possible operating modes are supported
 - Asynchronous mode with consistency locking
 - Asynchronous mode without consistency locking
 - Bus synchronous mode
 - Program synchronous mode
- Standard function bit commands
- Mailbox commands
- Reading and writing process data
- Reading diagnostic information

The TPMC821-SW-65 device driver supports the modules listed below:

TPMC821	INTERBUS Master G4	(PMC)
---------	--------------------	-------

To get more information about the features and use of TPMC821 devices it is recommended to read the manuals listed below.

TPMC821 User Manual
TEWS Windows Driver Installation Guide
IBS User Manuals

2 TPMC821 Device Driver Programming

The TPMC821-SW-65 Windows WDM device driver is a kernel mode device driver.

The standard file and device (I/O) functions (CreateFile, CloseHandle, and DeviceIoControl) provide the basic interface for opening and closing a resource handle and for performing device I/O control operations.

All of these standard Win32 functions are described in detail in the Windows Platform SDK Documentation (Windows base services / Hardware / Device Input and Output).

For details refer to the Win32 Programmers Reference of your used programming tools (C++, Visual Basic etc.)

2.1 TPMC821 Files and I/O Functions

The following section doesn't contain a full description of the Win32 functions for interaction with the TPMC821 device driver. Only the required parameters are described in detail.

2.1.1 Opening a TPMC821 Device

Before you can perform any I/O the TPMC821 device must be opened by invoking the CreateFile function. CreateFile returns a handle that can be used to access the TPMC821 device.

SYNOPSIS

```
HANDLE CreateFile(
    LPCTSTR lpFileName,
    DWORD   dwDesiredAccess,
    DWORD   dwShareMode,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    DWORD   dwCreationDistribution,
    DWORD   dwFlagsAndAttributes,
    HANDLE  hTemplateFile
)
```

PARAMETERS

lpFileName

Points to a null-terminated string, which specifies the name of the TPMC821 to open. The *lpFileName* string should be of the form \\.\TPMC821_x to open the device x. The ending x is a one-based number. The first device found by the driver is \\.\TPMC821_1, the second \\.\TPMC821_2 and so on.

dwDesiredAccess

Specifies the type of access to the TPMC821.
For the TPMC821 this parameter must be set to read-write access (GENERIC_READ | GENERIC_WRITE)

dwShareMode

Set of bit flags that specify how the object can be shared. Set to 0.

lpSecurityAttributes

This argument is a pointer to a security structure. Set to NULL for TPMC821 devices.

dwCreationDistribution

Specifies which action to take on files that exist, and which action to take when files do not exist. TPMC821 devices must be always opened OPEN_EXISTING.

dwFlagsAndAttributes

Specifies the file attributes and flags for the file. This value must be set to 0 (no overlapped I/O).

hTemplateFile

This value must be NULL for TPMC821 devices.

RETURN VALUE

If the function succeeds, the return value is an open handle to the specified TPMC821 device. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call *GetLastError*.

EXAMPLE

```
HANDLE    hDevice;

hDevice = CreateFile(
    "\\.\TPMC821_1",
    GENERIC_READ | GENERIC_WRITE,
    0,
    NULL,                // no security attrs
    OPEN_EXISTING,       // TPMC821 device always open existing
    0,                   // no overlapped I/O
    NULL
);

if (hDevice == INVALID_HANDLE_VALUE) {
    ErrorHandler( "Could not open device" ); // process error
}
```

SEE ALSO

CloseHandle(), Win32 documentation CreateFile()

2.1.2 Closing a TPMC821 Device

The CloseHandle function closes an open TPMC821 handle.

SYNOPSIS

```
BOOL CloseHandle(  
    HANDLE    hDevice;  
)
```

PARAMETERS

hDevice

Identifies an open TPMC821 handle.

RETURN VALUE

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

EXAMPLE

```
HANDLE    hDevice;  
  
if ( !CloseHandle( hDevice ) ) {  
    ErrorHandler( "Could not close device" ); // process error  
}
```

SEE ALSO

CreateFile(), Win32 documentation CloseHandle()

2.1.3 TPMC821 Device I/O Control Functions

The DeviceIoControl function sends a control code directly to a specified device driver, causing the corresponding device to perform the specified operation.

SYNOPSIS

```

BOOL DeviceIoControl(
    HANDLE          hDevice,
    DWORD           dwIoControlCode,
    LPVOID          lpInBuffer,
    DWORD           nInBufferSize,
    LPVOID          lpOutBuffer,
    DWORD           nOutBufferSize,
    LPDWORD          lpBytesReturned,
    LPOVERLAPPED    lpOverlapped
);

```

PARAMETERS

hDevice

Handle to the TPMC821 that is to perform the operation.

dwIoControlCode

Specifies the control code for the desired operation. This value identifies the specific operation to be performed. The following values are defined in *tpmc821.h*:

Value	Meaning
IOCTL_TP821_READ	Read process data out of the "DTA IN" area
IOCTL_TP821_WRITE	Write new process data to the "DTA OUT" area
IOCTL_TP821_BIT_CMD	Execute a standard function bit command
IOCTL_TP821_MBX_CMD	Execute a mailbox command and wait for confirmation
IOCTL_TP821_MBX_CMD_NOWAIT	Execute a mailbox command without waiting for confirmation
IOCTL_TP821_GET_DIAG	Get diagnostic information from the device
IOCTL_TP821_CONFIG	Configure the device driver
IOCTL_TP821_SET_HOST_FAIL	Set a serious host system failure interrupt
IOCTL_TP821_RESET_HOST_FAIL	Reset the host system failure interrupt
IOCTL_TP821_RESET_HARDWARE_FAIL	Reset the device hardware failure flag
IOCTL_TP821_MOD_INFO	Get information about the module (PCI-location, board variant)

See below for more detailed information on each control code.

To use these TPMC821 specific control codes the header file `tpmc821.h` must be included.

lpInBuffer

Pointer to a buffer that contains the data required to perform the operation.

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

lpOutBuffer

Pointer to a buffer that receives the operation's output data.

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

lpBytesReturned

Pointer to a variable that receives the size, in bytes, of the data stored into the buffer pointed to by *lpOutBuffer*. A valid pointer is required.

lpOverlapped

Pointer to an *Overlapped* structure. This value must be set to NULL (no overlapped I/O).

RETURN VALUE

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call `GetLastError`.

SEE ALSO

Win32 documentation `DeviceIoControl()`

2.1.3.1 IOCTL_TP821_READ

This TPCM821 control function reads actual process data out of the *DTA IN* area. A pointer to the caller's data buffer is passed by the parameters *lpInBuffer* and *lpOutBuffer* to the driver. This buffer contains variable length segments of data type *TP821_SEGMENT*. Each segment holds an exact description of the embedded data like data type, number of data items and offset in the *DTA IN* area. On entrance of this control function, every segment contains a description of the data items to read; on exit the driver fills the data union with the desired process data.

This relative complex mechanism has two advantages. First you can pick up occasional placed data items without copying the whole *DAT IN* buffer and second, word and long word organized data items can be automatically byte swapped by the driver. Remember Intel x86 CPU's use little-endian and Motorola respective the INTERBUS use big-endian alignment of data words.

```
typedef struct {
    USHORT      ItemNumber;
    USHORT      ItemType;
    USHORT      DataOffset;
    union {
        UCHAR    byte[1];
        USHORT    word[1];
        ULONG     lword[1];
    } u;
} TP821_SEGMENT, *PTP821_SEGMENT;
```

MEMBERS

ItemNumber

Specifies the number of items of the specified type in the data array. In other words it specifies the size of the array *u.byte[]*, *u.word[]* or *u.lword[]*.

ItemType

Specifies the data type of the embedded process data. Note, every data item in the segment must have the same type.

Type	Description
TP821_END	Specifies the last segment in the list. No data follows.
TP821_BYTE	Specifies a segment with byte data. The union part <i>byte[]</i> will be used.
TP821_WORD	Specifies a segment with word data. The union part <i>word[]</i> will be used and all words of the array will be byte swapped.
TP821_LWORD	Specifies a segment with long word data. The union part <i>lword[]</i> will be used and all long words will be byte swapped.

DataOffset

Specifies a byte offset from the beginning of the *DTA IN* area. The driver start reading data items from this offset and stores the requested number of items in the data union of the segment structure.

u

The union *u* contains three arrays. The size of these dynamic expandable arrays depends on the number of data items to read. Because the size of this arrays is only well-known at run-time you should never use the *sizeof()* function to determine the size of the segment structure.

The macro *SEGMENT_SIZE(pSeg)* (defined in *tpmc821.h*) delivers the correct structure size. The macro *NEXT_SEGMENT(pSeg)* (also defined in *tpmc821.h*) calculates a pointer to the begin of the following segment in the buffer. Both macros in combination should be used to assemble a read data buffer for the desired read request. The end of the read buffer is specified by a segment with type of *TP821_END*.

Please refer to the next example to see how to assemble a correct read buffer.

EXAMPLE

```
#include "tpmc821.h"

HANDLE          hDevice;
BOOLEAN         success;
ULONG           NumBytes, size;
UCHAR           SegmentBuffer[100];
PT821_SEGMENT   pSeg;

// [1]
size = 0;
pSeg = (PT821_SEGMENT)&SegmentBuffer;

// [2]
pSeg->ItemType    = TP821_BYTE;
pSeg->ItemNumber   = 4;
pSeg->DataOffset   = 0;
size += SEGMENT_SIZE(pSeg);           // add size of this segment

// [3]
pSeg = PNEXT_SEGMENT(pSeg);
pSeg->ItemType     = TP821_WORD;       // same data read as word
pSeg->ItemNumber    = 2;
pSeg->DataOffset    = 0;
size += SEGMENT_SIZE(pSeg);

pSeg = PNEXT_SEGMENT(pSeg);           // same data read as longword
pSeg->ItemType      = TP821_LWORD;
pSeg->ItemNumber     = 1;
pSeg->DataOffset     = 0;
size += SEGMENT_SIZE(pSeg);
```

```
// [4]
pSeg = PNEXT_SEGMENT(pSeg);          // End segment
pSeg->ItemType      = TP821_END;
pSeg->ItemNumber    = 0;
pSeg->DataOffset    = 0;
size += SEGMENT_SIZE(pSeg);

// [5]
success = DeviceIoControl (
    hDevice,          // TPMC821 handle
    IOCTL_TP821_READ, // control code
    &SegmentBuffer,    // template of data segments to read
    size,              // size of data segments
    &SegmentBuffer,    // filled data segments
    size,              // same size as the template buffer
    &NumBytes,         // number of bytes transferred
    NULL
);

if( success ) {
    /* Process data */
}
else {
    ErrorHandler ( "Device I/O control error" ); // process error
}
```

This example read the first four bytes of the *DTA IN* area within three segments with different types (byte, word and longword). If the first 4 bytes of the *DTA IN* area contains significant values you can realize the effect of byte swapping words and longwords (see also *tpmc821exa.c*).

- [1] After opening the device, the variable *size* is initialized with 0 and the segment pointer is set to begin of the segment buffer. Be sure that the size of the buffer is large enough to hold all segments.
- [2] The first segment contains 4 bytes read from offset 0 of the *DTA IN* area. After initializing of the segment we update the buffer size using the *SEGMENT_SIZE* macro. Do not use *sizeof()* instead.
- [3] Before initializing the next segment we calculate a new segment pointer using the *PNEXT_SEGMENT* macro. This macro simply adds the size of the previous segment to the previous segment pointer and returns the new segment pointer. The new segment starts without a gap direct after the previous segment.
- [4] The end of the segment list is specified by a segment with item type *TP821_END*. If this segment is missing the read request fails. Be sure that the size of the end segment is included in the total size of the segment list.
- [5] The *DeviceIoControl()* call transfers the request to the driver. The driver interprets the segment list and fills the corresponding data arrays with new process data.

2.1.3.2 IOCTL_TP821_WRITE

This TPCM821 control function writes new data to the *DTA OUT* area. A pointer to the callers data buffer is passed by the parameters *lpInBuffer* to driver. This buffer contains variable length segments of data type *TP821_SEGMENT*. Each segment holds an exact description of the embedded data like data type, number of data items, offset in the *DTA OUT* area and the data items to write.

This relative complex mechanism has two advantages. First you can write occasional placed data items without writing data in the whole *DAT OUT* buffer and second, word and long word organized data items can be automatically byte swapped by the driver. Remember Intel x86 CPU's use little-endian and Motorola respective the INTERBUS use big-endian alignment of data words.

```
typedef struct {
    USHORT      ItemNumber;
    USHORT      ItemType;
    USHORT      DataOffset;
    union {
        UCHAR      byte[1];
        USHORT      word[1];
        ULONG      lword[1];
    } u;
} TP821_SEGMENT, *PTP821_SEGMENT;
```

MEMBERS

ItemNumber

Specifies the number of items of the specified type in the data array. In other words it specifies the size of the array *u.byte[]*, *u.word[]* or *u.lword[]*.

ItemType

Specifies the data type of the embedded process data. Note, every data item in the segment must have the same type.

Type	Description
TP821_END	Specifies the last segment in the list. No data follows.
TP821_BYTE	Specifies a segment with byte data. The union part <i>byte[]</i> will be used.
TP821_WORD	Specifies a segment with word data. The union part <i>word[]</i> will be used and all words of the array will be byte swapped.
TP821_LWORD	Specifies a segment with long word data. The union part <i>lword[]</i> will be used and all long words will be byte swapped.

DataOffset

Specifies a byte offset from the beginning of the *DTA OUT* area where the new data should be written.

u

The union *u* contains three arrays. The size of these dynamic expandable arrays depends on the number of data items to write. Because the size of this arrays is only well-known at run-time you should never use the *sizeof()* function to determine the size of the segment structure.

The macro *SEGMENT_SIZE(pSeg)* (defined in *tpmc821.h*) delivers the correct structure size. The macro *NEXT_SEGMENT(pSeg)* (also defined in *tpmc821.h*) calculates a pointer to the begin of the following segment in the buffer. Both macros in combination should be used to assemble a write data buffer for the desired write request. The end of the write buffer is specified by a segment with type of *TP821_END*.

Please refer to the next example to see how to assemble a correct write buffer.

EXAMPLE

```
#include "tpmc821.h"
```

```
HANDLE          hDevice;
BOOLEAN         success;
ULONG           NumBytes, size;
UCHAR           SegmentBuffer[100];
PT821_SEGMENT   pSeg;

// [1]
size = 0;
pSeg = (PT821_SEGMENT)&SegmentBuffer;

// [2]
pSeg->ItemType    = TP821_BYTE;
pSeg->ItemNumber   = 4;
pSeg->DataOffset   = 0;
pSeg->u.byte[0]    = 1;
pSeg->u.byte[1]    = 2;
pSeg->u.byte[2]    = 3;
pSeg->u.byte[3]    = 4;
size += SEGMENT_SIZE(pSeg);           // add size of this segment

// [3]
pSeg = PNEXT_SEGMENT(pSeg);           // calculate next pointer
pSeg->ItemType     = TP821_LWORD;
pSeg->ItemNumber   = 1;
pSeg->DataOffset   = 4;
pSeg->u.lword[0]   = 0xAA55BB66;
size += SEGMENT_SIZE(pSeg);
```

```
// [4]
pSeg = PNEXT_SEGMENT(pSeg);          // End segment
pSeg->ItemType      = TP821_END;
pSeg->ItemNumber     = 0;
pSeg->DataOffset     = 0;
size += SEGMENT_SIZE(pSeg);

// [5]
success = DeviceIoControl (
    hDevice,                // TPMC821 handle
    IOCTL_TP821_WRITE,      // control code
    &SegmentBuffer,          // data segments to write
    size,                   // size of data segments
    NULL,
    0,
    &NumBytes,              // not used
    NULL
);

if( !success ) {
    ErrorHandler ( "Device I/O control error" ); // process error
}
```

This example does the following (see also `tpmc821exa.c`).

- [1] The variable *size* is initialized with 0 and the segment pointer is set to the beginning of the segment buffer. Be sure that the size of the buffer is large enough to hold all segments.
- [2] The first segment contains 4 bytes to write from offset 0 of the *DTA OUT* area. After initializing of the segment we update the buffer size using the *SEGMENT_SIZE* macro. Do not use *sizeof()* instead.
- [3] Before initializing the next segment we calculate a new segment pointer using the *PNEXT_SEGMENT* macro. This macro simply adds the size of the previous segment to the previous segment pointer and returns the new segment pointer. The new segment starts without a gap direct after the previous segment. The long word data item will be byte-swapped before writing to the *DTA OUT* area.
- [4] The end of the segment list is specified by a segment with item type *TP821_END*. If this segment is missing the read request fails. Be sure that the size of the end segment is included in the total size of the segment list.
- [5] The `DeviceIoControl()` call transfers the request to the driver. The driver interprets the segment list and writes the contents of the data array to the specified locations in the *DTA OUT* area.

The following example displays the memory layout of the segment buffer and the *DTA OUT* area after a successful write operation.

Segment values:

1st segment:

ItemNumber: 4
Item Type: TP821_BYTE
ItemOffset: 0x0
data: 0x01, 0x02, 0x03, 0x04

2nd segment:

ItemNumber: 1
Item Type: TP821_LWORD
ItemOffset: 0x0
data: 0xAA55BB66

End segment:

ItemNumber: 0
Item Type: TP821_END
ItemOffset: 0x0
data: (none)

The segment buffer has the following layout:

Offset	+0	+1	+2	+3	+4	+5	+6	+7
+0x00	0x04	0x00	0x01	0x00	0x00	0x00	0x01	0x02
+0x08	0x03	0x04	0x01	0x00	0x04	0x00	0x04	0x00
+0x10	0x66	0xBB	0x55	0xAA	0x00	0x00	0x00	0x00
+0x18	0x00	0x00	x	x	x	x	x	x

The *DTA OUT* area of the TPMC821 (after writing):

Offset	+0	+1	+2	+3	+4	+5	+6	+7
+0x00	0x01	0x02	0x03	0x04	0xAA	0x55	0xBB	0x66
+0x08	x	x	x	x	x	x	x	x

2.1.3.3 IOCTL_TP821_BIT_CMD

This control function allows the execution of various frequently used commands and command sequences without using mailbox commands. A pointer to the caller's parameter buffer is passed by the parameters *lpInBuffer* to driver.

Usually this kind of command execution is used on bit oriented host system.

```
typedef struct {
    USHORT      FunctionBit;
    USHORT      FunctionParam;
} TP821_BITCMD, *PTP821_BITCMD;
```

MEMBERS

FunctionBit

Specifies the bit number [0...6] of the standard function to execute.

FunctionParam

Specifies an optional parameter for the standard function.

Additional information about standard function bits and parameter values can be found in the User Manual – *INTERBUS Generation 4 Master Board*.

EXAMPLE

```
#include "tpmc821.h"

HANDLE      hDevice;
BOOLEAN     success;
TP821_BITCMD BitCmd;

BitCmd.FunctionBit      = 1<<0;    // Start_Data_Transfer_Req
BitCmd.FunctionParam    = 0;        // none

success = DeviceIoControl (
    hDevice,                // TPMC821 handle
    IOCTL_TP821_BIT_CMD,    // control code
    &BitCmd,
    sizeof(TP821_BITCMD),
    NULL,
    0,
    &NumBytes,              // not used
    NULL
);
```

2.1.3.4 IOCTL_TP821_MBX_CMD, IOCTL_TP821_MBX_CMD_NOWAIT

This control function is used to transmit messages from the host system to the IBS master (SSGI box 0). If an answer message is expected the received message (SSGI box 1) is copied direct to the user receive buffer. A pointer to the users transmit buffer is passed by the parameter *lpInBuffer* to driver. The parameter *lpOutBuffer* contains a pointer to the user receive buffer.

Transmit and receive buffer are organized as follows (valid for all services):

Word 1	Service_Code
Word 2	Parameter_Count
Word 3	Parameter
Word 4	Parameter
...	...
	Parameter

The control function *IOCTL_TP821_MBX_CMD_NOWAIT* returns immediately to the caller without waiting for an answer. This control function is used only for reset and unconfirmed PCP services.

Additional information about supported services can be found in the IBS User Manuals.

EXAMPLE

```
#include "tpmc821.h"

#define MAX_NUM_WORDS 100

HANDLE    hDevice;
BOOLEAN   success;
USHORT    RequestPar[MAX_NUM_WORDS];
USHORT    ResultPar[MAX_NUM_WORDS];

RequestPar[0] = 0x0710;           // Create Configuration Service
RequestPar[1] = 1;                // one parameter follow
RequestPar[2] = 1;                // number of frames to generate

success = DeviceIoControl (
    hDevice,                      // TPMC821 handle
    IOCTL_TP821_MBX_CMD,          // control code
    RequestPar,
    6,                            // size in bytes not words!
    ResultPar,
    MAX_NUM_WORDS * sizeof(USHORT),
    &NumBytes,                    // number of bytes returned
    NULL
);
```

2.1.3.5 IOCTL_TP821_GET_DIAG

This control function returns a structure with various diagnostic information to the caller. A pointer to the caller's diagnostic structure is passed by the parameters *lpOutBuffer* to driver.

```
typedef struct {
    USHORT        SysfailReg;
    USHORT        ConfigReg;
    USHORT        DiagReg;
    BOOLEAN        HardwareFailure;
    BOOLEAN        InitComplete;
} TP821_DIAG, *PTP821_DIAG;
```

MEMBERS

SysfailReg, ConfigReg, DiagReg

Returns the actual values of the corresponding hardware register in the coupling memory: *Status Sysfail Register, Configuration Register and Master Diagnostic Status Register*. The meaning of every bit in these registers is described in the User Manual – *INTERBUS Generation 4 Master Board*.

HardwareFailure

If the content is TRUE the IBS master has detected a hardware error. In this case the driver will not accept data transfer or message box commands until this state is left by the *IOCTL_TP821_RESET_HARDWARE_FAIL* command.

Note. A hardware failure could also occur after execution of the mailbox command *Reset_Controller_Board*.

InitComplete

This parameter is TRUE if the INTERBUS firmware has completed initialization.

EXAMPLE

```
#include "tpmc821.h"
```

```
HANDLE        hDevice;
BOOLEAN        success;
TP821_DIAG     DiagInfo;
```

```
...
```

```
...

success = DeviceIoControl (
    hDevice,                                // TPMC821 handle
    IOCTL_TP821_GET_DIAG,                   // control code
    NULL,
    0,
    &DiagInfo,
    sizeof(TP821_DIAG),
    &NumBytes,                              // number of bytes returned
    NULL
);

if( success ) {

    printf( "\nRead Diagnostic Information successful\n" );
    printf( "Status Sysfail Register      : %04Xhex\n",
        DiagInfo.SysfailReg );
    printf( "Configuration Register       : %04Xhex\n",
        DiagInfo.ConfigReg );
    printf( "Master Diagnostic Register : %04Xhex\n",
        DiagInfo.DiagReg );
    printf( "Hardware Failure              : %s\n",
        DiagInfo.HardwareFailure ? "TRUE" : "FALSE" );
    printf( "Initialization done           : %s\n",
        DiagInfo.InitComplete ? "TRUE" : "FALSE" );
}
else {
    ErrorHandler ( "Device I/O control error" ); // process error
}
```

2.1.3.6 IOCTL_TP821_CONFIG

This control function announces a new operating mode to the driver and change timeout values for mailbox and data transfer functions. Every time the host changes the operating mode (SetValue mailbox message) the driver must be introduced about that so he can handle following data transfer message in the right manner (see also *Automatic Configuration* in the example application).

A pointer to the callers configuration structure is passed by the parameters *lpInBuffer* to the driver.

```
typedef struct {
    USHORT      OperatingMode;
    ULONG       DataTimeout;
    ULONG       MailboxTimeout;
} TP821_CONFIG, *PTP821_CONFIG;
```

MEMBERS

OperationMode

Specifies the new operating mode. Valid operating modes are:

Operating Mode	Description
TP821_ASYNC	In asynchronous operating mode, the process data is updated by the INTERBUS firmware synchronously with the INTERBUS data cycles, but asynchronously with hosts' access to the process image. This operating mode is default after RESET.
TP821_ASYNC_LOCK	In this asynchronous operating mode the hosts' access to the process data is locked for reading and writing consistent data.
TP821_BUS_SYNC	Bus synchronous operating mode
TP821_PROG_SYNC	Program synchronous operating mode

Additional information about operating modes can be found in the User Manual – *INTERBUS Generation 4 Master Board*.

DataTimeout

Specifies a new timeout value for all following read and write commands from and to the *DTA IN* and *DTA OUT* area. The default timeout value is 2 seconds.

MailboxTimeout

Specifies a new timeout value for all following mailbox and function bit commands. The default timeout value is 10 seconds.

EXAMPLE

```
#include "tpmc821.h"

HANDLE          hDevice;
BOOLEAN         success;
TP821_CONFIG    ConfigPar;

// Setup new operating mode in the IBS firmware ...

ConfigPar.OperatingMode    = TP821_ASYNC_LOCK;
ConfigPar.DataTimeout      = 1;
ConfigPar.MailBoxTimeout   = 20;

success = DeviceIoControl (
    hDevice,                      // TPM821 handle
    IOCTL_TP821_CONFIG,          // control code
    &ConfigPar,
    sizeof(TP821_CONFIG),
    NULL,
    0,
    &NumBytes,                   // not used
    NULL
);

if( !success ) {
    ErrorHandler ( "Device I/O control error" ); // process error
}
```

2.1.3.7 IOCTL_TP821_SET_HOST_FAIL

This control function signals a serious host system failure to the TPMC821. On assertion of this host fail interrupt the TPMC821 resets all INTERBUS outputs and switch on the HF LED on the TPMC821 control panel.

If the driver was terminated the host system failure is automatically set by the driver.

EXAMPLE

```
#include "tpmc821.h"

HANDLE    hDevice;
BOOLEAN    success;

success = DeviceIoControl (
    hDevice,                // TPMC821 handle
    IOCTL_TP821_SET_HOST_FAIL, // control code
    NULL,
    0,
    NULL,
    0,
    &NumBytes,              // not used
    NULL
);

if( !success ) {
    ErrorHandler ( "Device I/O control error" ); // process error
}
```

2.1.3.8 IOCTL_TP821_RESET_HOST_FAIL

This control function resets the host system failure state in the TPMC821. No parameters are needed for execution of this control function.

EXAMPLE

```
#include "tpmc821.h"

HANDLE    hDevice;
BOOLEAN   success;

success = DeviceIoControl (
    hDevice,                      // TPMC821 handle
    IOCTL_TP821_RESET_HOST_FAIL, // control code
    NULL,
    0,
    NULL,
    0,
    &NumBytes,                    // not used
    NULL
);

if( !success ) {
    ErrorHandler ( "Device I/O control error" ); // process error
}
```


2.1.3.9 IOCTL_TP821_RESET_HARDWARE_FAIL

This control function resets the hardware failure flag in the device driver. The hardware failure flag was set after reception of a service interrupt request from the TPMC821. No parameters are needed for execution of this control function.

Additional information about the service interrupt request can be found in the TPCM821 User Manual and User Manuals for the INTERBUS Generation 4 Master Board which is part of the TPMC821 Engineering Manual.

EXAMPLE

```
#include "tpmc821.h"

HANDLE    hDevice;
BOOLEAN   success;

success = DeviceIoControl (
    hDevice,                                // TPMC821 handle
    IOCTL_TP821_RESET_HARDWARE_FAIL,       // control code
    NULL,
    0,
    NULL,
    0,
    &NumBytes,                             // not used
    NULL
);

if( !success ) {
    ErrorHandler ( "Device I/O control error" ); // process error
}
```

2.1.3.10 IOCTL_TP821_MOD_INFO

This control function returns information about the TPMC821 module. The returned information may be used to identify the location the module is mounted to. A pointer to the callers module info structure is passed by the parameters *lpOutBuffer* to driver.

```
typedef struct {
    UINT32 Variant
    UINT32 PciBusNo
    UINT32 PciDevNo;
} TP821_INFO_BUFFER, *PTP821_INFO_BUFFER;
```

MEMBERS

Variant

Returns the module variant. The value should always be 10. (TPMC821-10)

PciBusNo

Returns the PCI-Bus number the TPMC821 is mounted to.

PciDevNo

Returns the PCI-Device number of the TPMC821.

EXAMPLE

```
#include "tpmc821.h"

HANDLE    hDevice;
BOOLEAN    success;

success = DeviceIoControl (
    hDevice,                                // TPMC821 handle
    IOCTL_TP821_MOD_INFO,                  // control code
    NULL,
    0,
    &ModuleInfo,
    sizeof(TP821_INFO_BUFFER),
    &NumBytes,                               // not used
    NULL
);

...
```

...

```
if( !success ) {  
    ErrorHandler ( "Device I/O control error" ); // process error  
}
```

```
printf("Module type   = TPMC821-%02d\n", ModuleInfo.Variant);  
printf("PCI bus       = %d\n", ModuleInfo.PciBusNo);  
printf("PCI device     = %d\n", ModuleInfo.PciDevNo);
```