The Embedded I/O Company

TEWS TECHNOLOGIES

CARRIER-SW-65

Windows 2000/XP Device Driver

IPAC-Carrier

Version 1.2.x

User Manual

Issue 1.2.1

June 2008

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

TEWS TECHNOLOGIES LLC
9190 Double Diamond Parkway,
Suite 127, Reno, NV 89521, USA
www.tews.com
Phone: +1 (775) 850 5830
Fax: +1 (775) 201 0347
e-mail: usasales@tews.com

CARRIER-SW-65

IPAC-Carrier

Windows 2000/XP Device Driver

Issue	Description	Date
1.0	First Issue	October 22, 2003
1.1	Description for Generic Driver added / File-list extended	December 18, 2003
1.2	Description for VME Support added, Win XP Installation Description added, Win89/Me Installation Description removed	May 26, 2004
1.1.3	Description of Installation modified	November 1, 2004
1.1.4	File list changed	July 13, 2005
1.1.5	List of supported modules added, file list changed	July 03, 2006
1.2.0	TAMC100 support added, New address TEWS LLC	May 23, 2008
1.2.1	Files moved to subdirectory	June 20, 2008

Table of Content

1 Introduction

IndustryPack (IPAC) carrier boards have different implementations of the system to IndustryPack bus bridge logic, different implementations of interrupt and error handling and so on. Also the different byte ordering (big-endian versus little-endian) of CPU boards will cause problems on accessing the IndustryPack I/O and memory spaces.

To simplify the implementation of IPAC device driver which work with any supported carrier board, TEWS TECHNOLOGIES has designed a software architecture that hides all of these carrier board differences under a well defined interface.

Basically the drivers are split into three layers. The first layer handles accesses to the IPAC-Carrier cards. Different drivers are needed for different carriers. (We have implemented drivers for TEWS, SBS Carriers and VME Carrier boards with a Universe Master). These drivers are configuring the carrier boards and provide a function interface for the second layer. The second layer creates a device for every installed IP-slot used or unused, all slots are using the same driver. The IPAC-slots are checked, if an IPAC is mounted and for all mounted IPACs functions are provided for the IPAC drivers. The last layer is the IPAC driver which is handling the function of the IPAC. The IPAC driver must use the IPAC-slot functions for hardware accesses.

The CARRIER-SW-65 supports the modules listed below:

TEWS TPCI100	Carrier for 2 IndustryPack® modules	(PCI)
TEWS TPCI200	Carrier for 4 IndustryPack® modules	(PCI)
TEWS TCP201	Carrier for 4 IndustryPack® modules	(compactPCI)
TEWS TCP211	Carrier for 2 IndustryPack® modules	(compactPCI)
TEWS TCP212	Carrier for 2 IndustryPack® modules	(compactPCI)
TEWS TCP213	Carrier for 2 IndustryPack® modules	(compactPCI)
TEWS TCP220	Carrier for 4 IndustryPack® modules	(compactPCI)
TEWS TAMC100	Carrier for 1 IndustryPack® module	(AMC)
SBS PCI40(B)	Carrier for 4 IndustryPack® modules	(PCI)
SBS PCI60	Carrier for 6 IndustryPack® modules	(PCI)
SBS cPCI100/200	Carrier for 2/4 IndustryPack® modules	(compactPCI)
Universe 2	All VME-bus IPAC carrier	(VME)

2 Installation

Following files are located in directory CARRIER-SW-65 on the distribution media:

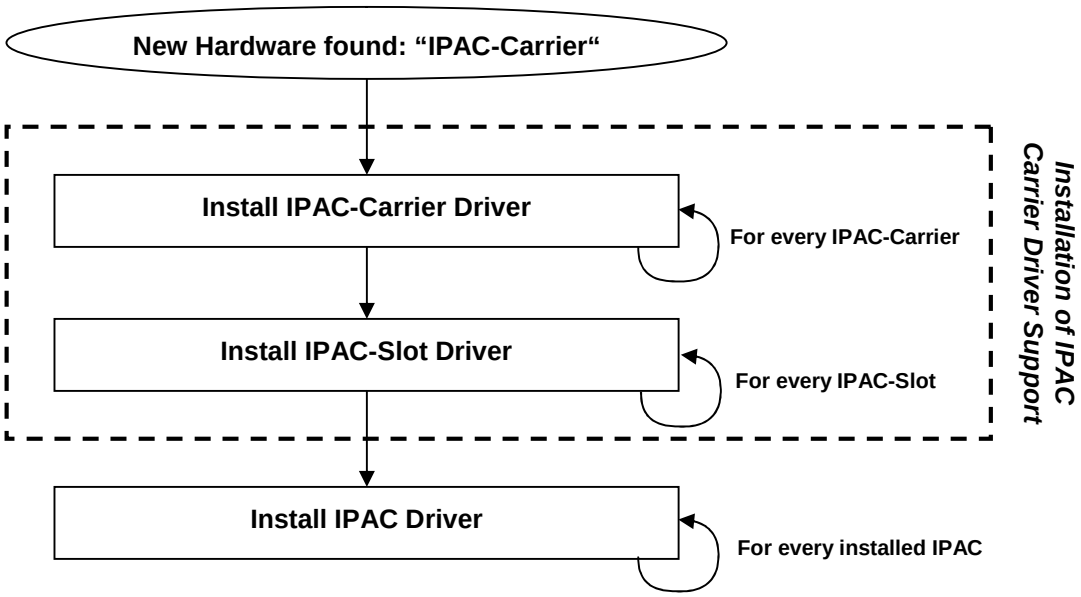
IPACBusFilter.sys	Device driver binary for IPAC Slot Driver
IPACBusFilter.inf	Installation script for IPAC Slot Driver
TEWSIPBus.sys	Device driver binary for TEWS Carrier Driver
TEWSIPBus.inf	Installation script for TEWS Carrier Driver
SBSIPBus.sys	Device driver binary for SBS Carrier Driver
SBSIPBus.inf	Installation script for SBS Carrier Driver
UVmeIpBus.sys	Device driver binary for Universe2 VME Carrier Driver
UVmeIpBus.inf	Installation script for Universe2 VME Carrier Driver
UVmeIpBus.reg	Registry configuration script for Universe2 VME Carrier Driver
genIPDrv.sys	Device driver binary for Generic IPAC Driver
genIPDrv.h	Application Include File for Generic IPAC Driver
genIPDrv.inf	Installation script for Generic IPAC Driver
Example/main.c	Example Application using the Generic IPAC Driver)
CARRIER-SW-65-1.2.1.pdf	This document
Release.txt	Release information
ChangeLog.txt	Release history

2.1 Software Installation

When installing the Universe VME carrier driver, be sure there is no other driver installed for the Universe PCI to VME Bridge.

2.1.1 General Installation Information

The Installation of the Carrier Driver Software has to be performed in two steps. Both steps must be done before the IPAC-Driver installation starts.



2.1.2 Windows 2000

This section describes how to install the IPAC-Carrier Device Drivers on a Windows 2000 operating system.

After installing the IPAC Carrier board(s) and boot-up your system, Windows 2000 setup will show a "***New hardware found***" dialog box.

The "***Upgrade Device Driver Wizard***" dialog box will appear on your screen.
Click "***Next***" button to continue.

In the following dialog box, choose "***Search for a suitable driver for my device***".
Click "***Next***" button to continue.

Insert the IPAC-Carrier driver disk; and select "***Disk Drive***" and/or "***CD-ROM***" in the dialog box.
Click "***Next***" button to continue.

Now the driver wizard should find a suitable device driver on the diskette.
Click "***Next***" button to continue.

Completing the upgrade device driver and click "***Finish***" to take all the changes effect.

Repeat the Instructions until drivers for all IPAC Carrier Boards are installed.

For VME Carrier we have to configure the resources now. (See *2.2 Configure VME-Carrier Driver*)

The next step to do is to install the IPAC Slot Drivers. Simply repeat the steps once more.

Now copy all needed files (CARRIER-SW-65.pdf) to the desired target directories.

After successful installation the IPAC-Carrier device driver will start immediately and create devices for all recognized carriers, IPAC-slots and mounted IPACs.

2.1.3 Windows XP

This section describes how to install the IPAC-Carrier Device Drivers on a Windows XP operating system.

After installing the IPAC Carrier board(s) and boot-up your system, Windows XP setup will show the search for new hardware window.
Insert the IPAC-Carrier driver disk and choose "***Automatic Software Install***".
Click "***Next***" button to continue.

A window will announce that the Windows-Logo test has failed.
Click "***continue install***" button to continue.

A window will announce that the driver has been installed.
Click "***Finish***" to take all the changes effect.

Repeat the Instructions until drivers for all IPAC Carrier Boards are installed.

For VME Carrier we have to configure the resources now. (See *2.2 Configure VME-Carrier Driver*)

The next step to do is to install the IPAC Slot Drivers. Simply repeat the steps before again, until all IPAC Slots are connected with a driver.

Now copy all needed files (CARRIER-SW-65.pdf) to the desired target directories.

After successful installation the IPAC-Carrier device driver will start immediately and create devices for all recognized carriers, IPAC-slots and mounted IPACs.

2.1.4 Confirming Windows 2000/XP Installation

To confirm that the driver has been properly loaded in Windows 2000/XP, perform the following steps:

From Windows 2000/XP, open the "***Control Panel***" from "My Computer".

Click the "***System***" icon and choose the "Hardware" tab, and then click the "***Device Manager***" button.

Click the "**+**" in front of "***Multifunction Devices***".
The drivers for the IPAC-Carrier Boards should appear and also an IPAC-Slot driver for each of the IPAC-Slots on the installed IPAC-Carrier Boards.

2.2 Configure VME-Carrier Driver

The VME-Bus doesn't support plug and play, so there must be a manual configuration for the VME IPAC slots. This configuration is done in the *UVmeIpBus.reg* file. This file splits into three parts: "*VMEbus Interface Configuration*", "*VMEbus Master Window Configuration*" and "VMEbus IPAC Slot Configuration"

After installation of the VME-Carrier Driver the *UVmeIpBus.reg* file must be installed to the system. Follow these steps to make the installation.

Modify *UVmeIpBus.reg* with a standard text editor.
Click right to the file and choose "***Merge***" from the context menu.
Restart the driver to complete the changes, or restart the system.

You will find the values in the windows registry in the following path and the subpathes:

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\UVmeIpBus]

You can also change values in the registry using *regedit*.

After every change of the registry values, the driver has to be restarted to get the new values.

2.2.1 VMEbus Interface Configuration

This part specifies the VME Controller Master setup.

For more detailed information of the values please refer to the VME-specification or the Universe manual.

The registry path is:

`[…\VMEbus]`

Default configuration:

```
"VMEbusRequestMode"=dword:00000000
"VMEbusReleaseMode"=dword:00000000
"VMEbusArbitrationMode"=dword:00000001
"VMEbusArbitrationTimeout"=dword:00000010
"VMEbusTimeout"=dword:00000040
"VMEbusRequestLevel"=dword:00000003
"NumberOfRetries"=dword:00000008
"PostedWriteTransferCount"=dword:00000200
"SYSCON"=dword:00000002
```

Description

VMEbusRequestMode

Value (hex)	Mode
0	Demand
1	Fair

VMEbusReleaseMode

Value (hex)	Mode
0	Release When Done (RWD)
1	Release on Request (ROR)

VMEbusArbitrationMode

Value (hex)	Mode
0	Round Robin
1	Priority

VMEbusArbitrationTimeout

Value (hex)	Timeout
0	0 µs
10	16 µs
100	256 µs

VMEbusTimeout

Value (hex)	Timeout
0	disabled
10	16 µs
20	32 µs
40	64 µs
80	128 µs
100	256 µs
200	512 µs
400	1024 µs

VMEbusRequestLevel

Value (hex)	Request Level
0	0
1	1
2	2
3	3

NumberOfRetries

Specifies the number of retries multiplied by 64. (0 – retry for ever, 1 – 64 retries, …) The value must be between 0h and Fh.

PostedWriteTransferCount

Value (hex)	Bytes
80	128
100	256
200	512
400	1024
800	2048
1000	4096

SYSCON

Value (hex)	System controller mode
0	NOT System Controller
1	System Controller
2	AUTO

2.2.2 VMEbus Master Window Configuration

This part specifies the Master Window configuration of the eight windows.

The registry path is:

`[…\VMEbus\Window`***n***`]`

n specifies the window number.

Default configuration:

```
[…\VMEbus\Window1]
; Window1 : A16/D16
"Enabled"=dword:00000001
"VMEbusBaseAddress"=dword:00000000
"WindowSize"=dword:00010000
"AddressModifier"=dword:00000029
"DataWidth"=dword:00000010

[…\VMEbus\Window2]
; Window1 : A24/D16
"Enabled"=dword:00000001
"VMEbusBaseAddress"=dword:00000000
"WindowSize"=dword:01000000
"AddressModifier"=dword:00000039
"DataWidth"=dword:00000010
```

Description

Enabled

Value (hex)	Description
0	Disable window
1	Enable window

VMEbusBaseAddress

VME-bus window start address

WindowSize

Window size in bytes

AddressModifier

Value (hex)	Access mode
9	A32 non-privileged data access
A	A32 non-privileged program access
D	A32 supervisory data access
E	A32 supervisory program access
29	A16 non-privileged access
2D	A16 supervisory access
39	A24 non-privileged data access
3A	A24 non-privileged program access
3D	A24 supervisory data access
3E	A24 supervisory program access

DataWidth

Value (hex)	Datawidth
8	8 bit
10	16 bit
2	32 bit

2.2.3 VMEbus IPAC Slot Configuration

This part specifies the IPAC Slot configuration of every IPAC slot on VME-bus. Each slot must get an own entry. These values mainly depend on the configuration of the VME IPAC carrier.

The registry path is:

`[…\VMEbus\Slot`***n***`]`

n specifies the slot number.

Default configuration:

```
[…\VMEbus\Slot1]

"Enabled"=dword:00000001

"ID_Address"=dword:00006080
"ID_Size"=dword:00000080
"ID_Window"=dword:00000001

"IO_Address"=dword:00006000
"IO_Size"=dword:00000080
"IO_Window"=dword:00000001

"MEM_Address"=dword:00D00000
"MEM_Size"=dword:00040000
"MEM_Window"=dword:00000002

"BaseVector"=dword:000000A0
"Level_INT0"=dword:00000001
"Level_INT1"=dword:00000002
```

Description

Enabled

Value (hex)	Description
0	Disable slot
1	Enable slot

ID_Address

Specifies the address offset of the ID space in the specified window.

ID_Size

Specifies the size of the ID space in bytes.

ID_Window

Specifies the window number the ID space is mapped to. The window number must be between 1 and 8. (see *2.2.2 VMEbus Master Window Configuration*)

IO_Address

Specifies the address offset of the I/O space in the specified window.

IO_Size

Specifies the size of the I/O space in bytes.

IO_Window

Specifies the window number the I/O space is mapped to. The window number must be between 1 and 8. (see *2.2.2 VMEbus Master Window Configuration*)

MEM_Address

Specifies the address offset of the memory space in the specified window.

MEM_Size

Specifies the size of the memory space in bytes.

MEM_Window

Specifies the window number the memory space is mapped to. The window number must be between 1 and 8. (see *2.2.2 VMEbus Master Window Configuration*)

BaseVector

Specifies the interrupt base vector for this IPAC slot, 8 vectors are reserved for each of the slots. Valid vectors are between 40h and F8h

Level_INT0

Specifies the VME interrupt level INT0 will generate on the VME Bus. Valid VME interrupt levels are between 1 and 7. (0 – no interrupt is generated on INT0)

Level_INT1

Specifies the VME interrupt level INT1 will generate on the VME Bus. Valid VME interrupt levels are between 1 and 7. (0 – no interrupt is generated on INT1)

3 Customer IPAC Carrier Support

If your IPAC carrier isn't supported by the carrier port drivers on the distribution diskette and your carrier board is a PCI bus carrier please contact TEWS TECHNOLOGIES.

Usually we will implement the carrier driver without any charge within a few days.

4 Generic IPAC Driver

The Generic Driver can be used for first steps using an IPAC. The driver allows access to the IPAC. The module interface (IP-Clockrate, Space size ...) can be configured by the application. Only interrupts are not implemented.

The Generic Driver allows following functions:

- writing bytes, shorts and longwords
- reading bytes, shorts and longwords
- configuring the IPAC-slot (allocate)
- unconfiguring the IPAC-slot (free)

4.1 Installation

4.1.1 Before Installation

To use the driver for a specific IPAC, copy the driver files to a local directory and modify the installation-file. The supported modules are identified by the hardware identifier. For the IPAC modules identified by the IPAC Carrier driver this will be 'IPACSlot\' followed by the IPAC Name. For undefined modules it will be 'IPACSlot\' followed by 'MAN<manufacturer ID>_MOD<model number> where manufacturer ID and model number are read from the IPACs ID-Prom.

Find the following line.

```
[TEWS.Mfg]
```

This line identifies a list of hardware identifiers, specifying the IPACs the driver will handle.

The TIP255 (identified) will be handled by the following example.

```
[TEWS.Mfg]
%TTG.SvcDesc% = TTG,IPACSlot\TIP255
```

If the TIP255 has not been identified the following example the following modification will handle it.

```
[TEWS.Mfg]
%TTG.SvcDesc% = TTG, IPACSlot\MANb3_MOD31
```

The following example shows an example if two modules shall be handled by the driver.

```
[TEWS.Mfg]
%TTG.SvcDesc% = TTG,IPACSlot\TIP255
%TTG.SvcDesc% = TTG,IPACSlot\MANb3_MOD30
```

4.1.2 Windows 2000

This section describes how to install the Generic IPAC Device Driver on a Windows 2000 operating system.

1. After installing the IPAC card(s) and boot-up your system, Windows 2000 setup will show a "***New hardware found***" dialog box.
2. The "***Upgrade Device Driver Wizard***" dialog box will appear on your screen.
   Click "***Next***" button to continue.
3. In the following dialog box, choose "***Search for a suitable driver for my device***".
   Click "***Next***" button to continue.
4. Select local path with the modified inf-File.
   Click "***Next***" button to continue.
5. Now the driver wizard should find a suitable device driver.
   Click "***Next***" button to continue.
6. Completing the upgrade device driver and click "***Finish***" to take all the changes effect.

After successful installation the Generic IPAC device driver will start immediately and creates devices (genIPDrv_1, genIPDrv_2, ...) for all recognized IPAC modules.

4.1.3 Confirming Windows 2000 Installation

To confirm that the driver has been properly loaded in Windows 2000, perform the following steps:

1. From Windows 2000, open the "***Control Panel***" from "***My Computer***".
2. Click the "***System***" icon and choose the "***Hardware***" tab, and then click the "***Device Manager***" button.
3. Click the "**+**" in front of "***Other Devices***".
   The driver "***genIPDrv***" should appear.

4.1.4 Windows 98 SE / Windows ME

This section describes how to install the Generic IPAC Device Driver on a Windows 98 Second Edition (SE) operating system.

1. After installing the IPAC card(s) and boot-up your system, Windows 98 SE setup will show a "***New hardware found***" dialog box.
2. The "***Add New Hardware Wizard***" dialog box will appear on your screen, informing you that it has found a new PCI device.
   Click "***Next***" button to continue.
3. In the following dialog box, choose "***Search for a better driver than the one your device is using now***".
   Click "***Next***" button to continue.
4. In the following dialog box, select "***Specify a location***".
5. Select local path with the modified inf-File.
   Click "***Next***" button to continue.

After successful installation the Generic IPAC device driver will start immediately and creates devices (genIPDrv_1, genIPDrv_2, ...) for all recognized IPAC modules.

4.1.5 Confirming Windows 98 SE / Windows ME Installation

To confirm that the driver has been properly loaded in Windows, perform the following steps:

1. Choose "***Settings***" from the "***Start***" menu.
2. Choose "***Control Panel***" and then double-click on the "***System***" icon.
3. Choose the "***Device Manager***" tab, and then click the "**+**" in front of "***Other Devices***".
   The driver "***genIPDrv***" should appear.

4.2 Generic IPAC Device Driver Programming

The Generic IPAC 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 device 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.)

4.2.1 IPAC Files and I/O Functions

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

4.2.1.1 Opening an IPAC Device

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

```
HANDLE CreateFile(
    LPCTSTR lpFileName,                          // pointer to filename
    DWORD dwDesiredAccess,                       // access (read-write) mode
    DWORD dwShareMode,                           // share mode
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,  // pointer to security attributes
    DWORD dwCreationDistribution,                // how to create
    DWORD dwFlagsAndAttributes,                  // file attributes
    HANDLE hTemplateFile                         // handle to file with attributes to copy
);
```

Parameters

lpFileName

Points to a null-terminated string that specifies the name of the IPAC to open.
The *lpFileName* string should be of the form **\\.\genIPDrv_*x*** to open the device *x*. The ending *x* is a one-based number. The first device found by the driver is \\.\genIPDrv_1, the second \\.\genIPDrv_2 and so on.

dwDesiredAccess

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

dwShareMode

A set of bit flags that specifies how the object can be shared for read and write. Unimportant for Generic IPAC, set to 0.

lpSecurityAttributes

Pointer to a security structure. Set to NULL for generic IPAC devices.

dwCreationDistribution

Specifies which action to take on files that exist and which action to take when files that do not exist. Generic IPAC 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 0 for Generic IPAC devices.

Return Value

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

Example

```
HANDLE    hDevice;

hDevice = CreateFile(
     “\\\\.\\genIPDrv_1”,
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,              // no security attrs
     OPEN_EXISTING,     // IPAC 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()

4.2.1.2 Closing an IPAC Device

The **CloseHandle** function closes an open IPAC handle.

```
BOOL CloseHandle(
      HANDLE hDevice;                          // handle to a IPAC device to close
);
```

Parameters

hDevice

Identifies an open IPAC 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;

hDevice = CreateFile(
     “\\\\.\\geiIPDrv_1”,
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,              // no security attrs
     OPEN_EXISTING,     // IPAC device always open existing
     0,                 // no overlapped I/O
     NULL
);
if(hDevice == INVALID_HANDLE_VALUE) {
 ErrorHandler("Could not open device"); // process error
}

/* ... do some device I/O ...  */

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

See Also

CreateFile(), Win32 documentation CloseHandle()

4.2.1.3 IPAC 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.

```
BOOL DeviceIoControl(
    HANDLE hDevice,                 // handle to device of interest
    DWORD dwIoControlCode,          // control code of operation to perform
    LPVOID lpInBuffer,              // pointer to buffer to supply input data
    DWORD nInBufferSize,            // size of input buffer
    LPVOID lpOutBuffer,             // pointer to buffer to receive output data
    DWORD nOutBufferSize,           // size of output buffer
    LPDWORD lpBytesReturned,        // pointer to variable to receive output byte count
    LPOVERLAPPED lpOverlapped       // pointer to overlapped structure for asynchronous
                                    // operation
);
```

Parameters

hDevice

Handle to the IPAC that is to perform the operation.

dwIoControlCode

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

Value	Meaning
GENIPDRV_CONFIGURE	Configure Carrier IPAC Slot and allocate IPAC addresses
GENIPDRV_UNCONFIGURE	Release IPAC addresses
GENIPDRV_READ_UCHAR	Get data from IPAC Device (8-bit accesses)
GENIPDRV_READ_USHORT	Get data from IPAC Device (16-bit accesses)
GENIPDRV_READ_ULONG	Get data from IPAC Device (32-bit accesses)
GENIPDRV_WRITE_UCHAR	Write data to IPAC Device (8-bit accesses)
GENIPDRV_WRITE_USHORT	Write data to IPAC Device (16-bit accesses)
GENIPDRV_WRITE_ULONG	Write data to IPAC Device (32-bit accesses)

See behind for more detailed information on each control code.

To use these Generic IPAC specific control codes the header file genIPDrv.h must be included in the application.

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()

4.2.1.4 GENIPDRV_CONFIGURE

The configure function allocates address spaces to access the IPAC and sets up the IPAC slot. This function must be used before any other access to the IPAC is done.

The parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the configuration buffer (*GENIPDRV_CONFIGURE_BUF*) to the device driver.

```
typedef struct _GENIPDRV_CONFIGURE_BUF
{
    BOOLEAN     enable8BitIP;   // TRUE - 8 Bit IP Bus
                                // FALSE - 16 Bit IP Bus    (startup)
    BOOLEAN     enable32MHz;    // TRUE - 32 MHz IP Clock
                                // FALSE - 8 MHz IP Clock   (startup)
    ULONG       sizeIDSpace;    // Size of ID-Space
    BOOLEAN     swapIDSpace;    // Little <-> Big Endian swapping enable
    ULONG       sizeIOSpace;    // Size of I/O-Space
    BOOLEAN     swapIOSpace;    // Little <-> Big Endian swapping enable
    ULONG       sizeMEMSpace;   // Size of Memory-Space
    BOOLEAN     swapMEMSpace;   // Little <-> Big Endian swapping enable
} GENIPDRV_CONFIGURE_BUF, *PGENIPDRV_CONFIGURE_BUF;
```

enable8BitIP

This parameter specifies the width of the IPAC bus. (If supported by hardware)

Value	Description
TRUE	The IPAC bus will be configured as 8-bit bus
FALSE	The IPAC bus will be configured as 16-bit bus

enable32MHz

This parameter specifies the IPAC clock speed. (If supported by hardware)

Value	Description
TRUE	The IPAC clock speed will be set to 32MHz
FALSE	The IPAC clock speed will be set to 8MHz

sizeIDSpace

This parameter must specify the needed size of the ID-space.

swapIDSpace

This parameter specifies if the data should be swapped for read and write accesses to ID-space.

Value	Description
TRUE	Data will be swapped
FALSE	Data will not be swapped

sizeIOSpace

This parameter must specify the needed size of the I/O-space.

swapIOSpace

This parameter specifies if the data should be swapped for read and write accesses to I/O-space.

Value	Description
TRUE	Data will be swapped
FALSE	Data will not be swapped

sizeMEMSpace

This parameter must specify the needed size of the Memory-space.

swapMEMSpace

This parameter specifies if the data should be swapped for read and write accesses to Memory-space.

Value	Description
TRUE	Data will be swapped
FALSE	Data will not be swapped

Example

```
#include                “genIPDrv.h”

GENIPDRV_CONFIGURE_BUF  configBuf;
HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;

...
```

```
...

// Configure IPAC slot:
//    - 16-bit bus width
//    - 32 MHz clock speed
//    - 32 Byte ID-Space (not swapped)
//    - 18 Byte I/O-Space (swapped)
//    - no Memory-Space
configBuf.enable8BitIP =    FALSE;
configBuf.enable32MHz =     TRUE;
configBuf.sizeIDSpace =     32;
configBuf.swapIDSpace =     FALSE;
configBuf.sizeIOSpace =     18;
configBuf.swapIOSpace =     TRUE;
configBuf.sizeMEMSpace =    0x0;
configBuf.swapMEMSpace =    FALSE;

//  Send request to the device driver
//
success = DeviceIoControl (
                  hCurrent,              // IPAC handle
                  GENIPDRV_CONFIGURE,    // control code
                  &configBuf,
                  sizeof(GENIPDRV_CONFIGURE_BUF),
                  &configBuf,
                  sizeof(GENIPDRV_CONFIGURE_BUF),
                  &NumBytes,             //  number of bytes transferred
                  NULL                   //  not over lapped
              );
if( success ) {
     // IPAC-Slot Configured
}
else {
     // IPAC-Slot Configuration failed
}
```

Error Codes

All returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl()

4.2.1.5 GENIPDRV_UNCONFIGURE

The unconfigure function releases address spaces allocater for IPAC access.

The parameter *lpInBuffer* and *lpOutBuffer* must pass a *NULL* pointer to the device driver.

Example

```
#include                “genIPDrv.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;

...

success = DeviceIoControl (
                  hCurrent,               // IPAC handle
                  GENIPDRV_UNCONFIGURE,   // control code
                  NULL,
                  0,
                  NULL,
                  0,
                  &NumBytes,              // number of bytes transferred
                  NULL                    // not over lapped
              );
//
// Check the result of the last device I/O control operation
//
if( success ) {
     // IPAC-Slot unconfigured
}
else {
     // IPAC-Slot Unconfiguration failed
}
```

Error Codes

All returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl()

4.2.1.6 GENIPDRV_READ_UCHAR

The read uchar function reads a buffer of 8-bit data from a specified address space. Before this function is used the GENIPDRV_CONFIGURE function must be called.

The parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the I/O buffer (*GENIPDRV_IO_BUF*) to the device driver.

```
typedef struct _GENIPDRV_IO_BUF
{
    UCHAR       space;                          // address space to read from
    ULONG       offset;                         // address offset in space
    ULONG       size;                           // number of uchar/ushort/ulong
    UCHAR       buffer[GENIPDRV_MAXIOBUF];  // pointer to buffer
} GENIPDRV_IO_BUF, *PGENIPDRV_IO_BUF;
```

space

This parameter specifies the address of the IPAC the data shall be read from.

Value	Description
GENIPDRV_IDSPACE	Read from ID Space
GENIPDRV_IOSPACE	Read from I/O space
GENIPDRV_MEMSPACE	Read from memory space

offset

This parameter specifies the starting offset in the selected space.

size

This parameter specifies the length of the buffer to read.

buffer[]

This array will be filled with the data read from the specified position. The size of the buffer can be changed by changing the value of *GENIPDRV_MAXIOBUF* in genIPDrv.h. The value is specified in byte.

Example

```
#include           “genIPDrv.h”

GENIPDRV_IO_BUF    ioBuf;
HANDLE             hDevice;
BOOLEAN            success;
ULONG              NumBytes;
PUCHAR             ucPtr;

...

// Read 16 Bytes from IPAC ID-Space starting at offset 0x10
ioBuf.space = GENIPDRV_IDSPACE;
ioBuf.offset =0x10;
ioBuf.size =  0x10;

//  Send request to the device driver
//
success = DeviceIoControl (
              hCurrent,                   // IPAC handle
              GENIPDRV_READ_UCHAR,        // control code
              &ioBuf,
              sizeof(GENIPDRV_IO_BUF),
              &ioBuf,
              sizeof(GENIPDRV_IO_BUF),
              &NumBytes,                  // number of bytes transferred
              NULL                        // not over lapped
          );
if( success ) {
     // read access OK
     ucPtr = &ioBuf.Buffer[0];    // Set pointer to data
}
else {
     // read access failed
}
```

Error Codes

All returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl()

4.2.1.7 GENIPDRV_READ_USHORT

The read ushort function reads a buffer of 16-bit data from a specified address space. Before this function is used the GENIPDRV_CONFIGURE function must be called.

The parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the I/O buffer (*GENIPDRV_IO_BUF*) to the device driver.

```
typedef struct _GENIPDRV_IO_BUF
{
    UCHAR       space;                          // address space to read from
    ULONG       offset;                         // address offset in space
    ULONG       size;                           // number of uchar/ushort/ulong
    UCHAR       buffer[GENIPDRV_MAXIOBUF];  // pointer to buffer
} GENIPDRV_IO_BUF, *PGENIPDRV_IO_BUF;
```

space

This parameter specifies the address of the IPAC the data shall be read from.

Value	Description
GENIPDRV_IDSPACE	Read from ID Space
GENIPDRV_IOSPACE	Read from I/O space
GENIPDRV_MEMSPACE	Read from memory space

offset

This parameter specifies the starting offset in the selected space.

size

This parameter specifies the number of words to read.

buffer[]

This array will be filled with the data read from the specified position. The size of the buffer can be changed by changing the value of *GENIPDRV_MAXIOBUF* in genIPDrv.h. The value is specified in byte.

Example

```
#include           “genIPDrv.h”

GENIPDRV_IO_BUF    ioBuf;
HANDLE             hDevice;
BOOLEAN            success;
ULONG              NumBytes;
PUSHORT            usPtr;

...

// Read 8 Words from IPAC I/O-Space starting at offset 0x10
ioBuf.space = GENIPDRV_IOSPACE;
ioBuf.offset =0x10;
ioBuf.size =  0x8;

//  Send request to the device driver
//
success = DeviceIoControl (
               hCurrent,                    // IPAC handle
               GENIPDRV_READ_USHORT,        // control code
               &ioBuf,
               sizeof(GENIPDRV_IO_BUF),
               &ioBuf,
               sizeof(GENIPDRV_IO_BUF),
               &NumBytes,                   // number of bytes transferred
               NULL                         // not over lapped
          );
if( success ) {
     // read access OK
     usPtr = (PUSHORT)&ioBuf.Buffer[0];    // Set pointer to data
}
else {
     // read access failed
}
```

Error Codes

All returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl()

4.2.1.8 GENIPDRV_READ_ULONG

The read ulong function reads a buffer of 32-bit data from a specified address space. Before this function is used the GENIPDRV_CONFIGURE function must be called.

The parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the I/O buffer (*GENIPDRV_IO_BUF*) to the device driver.

```
typedef struct _GENIPDRV_IO_BUF
{
    UCHAR       space;                          // address space to read from
    ULONG       offset;                         // address offset in space
    ULONG       size;                           // number of uchar/ushort/ulong
    UCHAR       buffer[GENIPDRV_MAXIOBUF];  // pointer to buffer
} GENIPDRV_IO_BUF, *PGENIPDRV_IO_BUF;
```

space

This parameter specifies the address of the IPAC the data shall be read from.

Value	Description
GENIPDRV_IDSPACE	Read from ID Space
GENIPDRV_IOSPACE	Read from I/O space
GENIPDRV_MEMSPACE	Read from memory space

offset

This parameter specifies the starting offset in the selected space.

size

This parameter specifies the number of longwords l to read.

buffer[]

This array will be filled with the data read from the specified position. The size of the buffer can be changed by changing the value of *GENIPDRV_MAXIOBUF* in genIPDrv.h. The value is specified in byte.

Example

```
#include           “genIPDrv.h”

GENIPDRV_IO_BUF    ioBuf;
HANDLE             hDevice;
BOOLEAN            success;
ULONG              NumBytes;
PULONG             ulPtr;

...

// Read 4 Longwords from IPAC Memory-Space starting at offset 0x10
ioBuf.space = GENIPDRV_MEMSPACE;
ioBuf.offset =0x10;
ioBuf.size =  0x4;

//  Send request to the device driver
//
success = DeviceIoControl (
               hCurrent,                    // IPAC handle
               GENIPDRV_READ_ULONG,         // control code
               &ioBuf,
               sizeof(GENIPDRV_IO_BUF),
               &ioBuf,
               sizeof(GENIPDRV_IO_BUF),
               &NumBytes,                   // number of bytes transferred
               NULL                         // not over lapped
          );
if( success ) {
     // read access OKO
     ulPtr = (PULONG)&ioBuf.Buffer[0];     // Set pointer to data
}
else {
     // read access failed
}
```

Error Codes

All returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl()

4.2.1.9 GENIPDRV_WRITE_UCHAR

The write uchar function writes a buffer of 8-bit data to a specified address space. Before this function is used the GENIPDRV_CONFIGURE function must be called.

The parameter *lpInBuffer* must pass a pointer to the I/O buffer (*GENIPDRV_IO_BUF*) to the device driver. The parameter *lpOutBuffer* must pass a *NULL* pointer to the device driver.

```
typedef struct _GENIPDRV_IO_BUF
{
    UCHAR       space;          // address space to read from
    ULONG       offset;         // address offset in space
    ULONG       size;           // number of uchar/ushort/ulong
    UCHAR       buffer[GENIPDRV_MAXIOBUF];  // pointer to buffer
} GENIPDRV_IO_BUF, *PGENIPDRV_IO_BUF;
```

space

This parameter specifies the address of the IPAC the data shall be read from.

Value	Description
GENIPDRV_IDSPACE	Read from ID Space
GENIPDRV_IOSPACE	Read from I/O space
GENIPDRV_MEMSPACE	Read from memory space

offset

This parameter specifies the starting offset in the selected space.

size

This parameter specifies the length of the buffer to write.

buffer[]

This array must be filled with the data to write to the specified position. The size of the buffer can be changed by changing the value of *GENIPDRV_MAXIOBUF* in genIPDrv.h. The value is specified in byte.

Example

```
#include          “genIPDrv.h”

GENIPDRV_IO_BUF   ioBuf;
HANDLE            hDevice;
BOOLEAN           success;
ULONG             NumBytes;
PUCHAR            ucPtr;

...
```

```
...

// Write 3 Bytes (0x11,0x22,0x33) to IPAC I/O-Space starting at offset 0x10
ioBuf.space = GENIPDRV_IOSPACE;
ioBuf.offset =0x10;
ioBuf.size =  0x3;

ucPtr = &ioBuf.Buffer[0];   // Set pointer to data
ucPtr[0] = 0x11;
ucPtr[1] = 0x22;
ucPtr[2] = 0x33;

//  Send request to the device driver
//
success = DeviceIoControl (
              hCurrent,                   // IPAC handle
              GENIPDRV_WRITE_UCHAR,       // control code
              &ioBuf,
              sizeof(GENIPDRV_IO_BUF),
              NULL,
              0,
              &NumBytes,                  // number of bytes transferred
              NULL                        // not over lapped
         );
if( success ) {
     // write access OK
}
else {
     // write access failed
}
```

Error Codes

All returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl()

4.2.1.10 GENIPDRV_WRITE_USHORT

The write ushort function writes a buffer of 16-bit data to a specified address space. Before this function is used the GENIPDRV_CONFIGURE function must be called.

The parameter *lpInBuffer* must pass a pointer to the I/O buffer (*GENIPDRV_IO_BUF*) to the device driver. The parameter *lpOutBuffer* must pass a *NULL* pointer to the device driver.

```
typedef struct _GENIPDRV_IO_BUF
{
    UCHAR       space;          // address space to read from
    ULONG       offset;         // address offset in space
    ULONG       size;           // number of uchar/ushort/ulong
    UCHAR       buffer[GENIPDRV_MAXIOBUF];  // pointer to buffer
} GENIPDRV_IO_BUF, *PGENIPDRV_IO_BUF;
```

space

This parameter specifies the address of the IPAC the data shall be read from.

Value	Description
GENIPDRV_IDSPACE	Read from ID Space
GENIPDRV_IOSPACE	Read from I/O space
GENIPDRV_MEMSPACE	Read from memory space

offset

This parameter specifies the starting offset in the selected space.

size

This parameter specifies number of words to write.

buffer[]

This array must be filled with the data to write to the specified position. The size of the buffer can be changed by changing the value of *GENIPDRV_MAXIOBUF* in genIPDrv.h. The value is specified in byte.

Example

```
#include          “genIPDrv.h”

GENIPDRV_IO_BUF   ioBuf;
HANDLE            hDevice;
BOOLEAN           success;
ULONG             NumBytes;
PUSHORT           usPtr;

...
```

```
...

// Write 3 Words (0x1111,0x2222,0x3333) to IPAC I/O-Space starting
// at offset 0x10
ioBuf.space = GENIPDRV_IOSPACE;
ioBuf.offset =0x10;
ioBuf.size =  0x3;

usPtr = (PUSHORT)&ioBuf.Buffer[0];   // Set pointer to data
usPtr[0] = 0x1111;
usPtr[1] = 0x2222;
usPtr[2] = 0x3333;

//  Send request to the device driver
//
success = DeviceIoControl (
              hCurrent,                    // IPAC handle
              GENIPDRV_WRITE_USHORT,       // control code
              &ioBuf,
              sizeof(GENIPDRV_IO_BUF),
              NULL,
              0,
              &NumBytes,                   // number of bytes transferred
              NULL                         // not over lapped
         );
if( success ) {
     // write access OK
}
else {
     // write access failed
}
```

Error Codes

All returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl()

4.2.1.11 GENIPDRV_WRITE_ULONG

The write ulong function writes a buffer of 32-bit data to a specified address space. Before this function is used the GENIPDRV_CONFIGURE function must be called.

The parameter *lpInBuffer* must pass a pointer to the I/O buffer (*GENIPDRV_IO_BUF*) to the device driver. The parameter *lpOutBuffer* must pass a *NULL* pointer to the device driver.

```
typedef struct _GENIPDRV_IO_BUF
{
    UCHAR       space;          // address space to read from
    ULONG       offset;         // address offset in space
    ULONG       size;           // number of uchar/ushort/ulong
    UCHAR       buffer[GENIPDRV_MAXIOBUF];  // pointer to buffer
} GENIPDRV_IO_BUF, *PGENIPDRV_IO_BUF;
```

space

This parameter specifies the address of the IPAC the data shall be read from.

Value	Description
GENIPDRV_IDSPACE	Read from ID Space
GENIPDRV_IOSPACE	Read from I/O space
GENIPDRV_MEMSPACE	Read from memory space

offset

This parameter specifies the starting offset in the selected space.

size

This parameter specifies the number of longwords to write.

buffer[]

This array must be filled with the data to write to the specified position. The size of the buffer can be changed by changing the value of *GENIPDRV_MAXIOBUF* in genIPDrv.h. The value is specified in byte.

Example

```
#include          “genIPDrv.h”

GENIPDRV_IO_BUF   ioBuf;
HANDLE            hDevice;
BOOLEAN           success;
ULONG             NumBytes;
PULONG            ulPtr;

...
```

```
...

// Write 2 Longwords (0x11111111,0x22222222) to IPAC Memory-Space
// starting at offset 0x10
ioBuf.space = GENIPDRV_MEMSPACE;
ioBuf.offset =0x10;
ioBuf.size =  0x2;

ulPtr = (PULONG)&ioBuf.Buffer[0];// Set pointer to data
ulPtr[0] = 0x11111111;
ulPtr[1] = 0x22222222;

//  Send request to the device driver
//
success = DeviceIoControl (
               hCurrent,                    // IPAC handle
               GENIPDRV_WRITE_ULONG,        // control code
               &ioBuf,
               sizeof(GENIPDRV_IO_BUF),
               NULL,
               0,
               &NumBytes,                   // number of bytes transferred
               NULL                         // not over lapped
          );
if( success ) {
     // write access OK
}
else {
     // write access failed
}
```

Error Codes

All returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl()