The Embedded I/O Company

TEWS TECHNOLOGIES

CARRIER-SW-65

Windows Device Driver

IPAC-Carrier

Version 2.3.x

User Manual

Issue 2.3.0

January 2014

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

CARRIER-SW-65

Windows Device Driver

IPAC-Carrier

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
1.3.0	Support of IPAC ID-PROM Type II (VITA4) added	October 17, 2008
1.4.0	Description of IPDrvCustom (Custom IPAC driver) added, TAMC200 support added	December 23, 2008
2.0.0	General driver and manual update, Windows 7 Support added, chapters of unsupported parts removed, address TEWS LLC removed	September 9, 2011
2.1.0	Installation for Windows 2000 modified, Support for TAMC220 added	October 19, 2011
2.2.0	Support for TPCE200 added	July 31, 2013
2.3.0	Generic IPAC Driver, Naming of carrier locations	January 3, 2013

Table of Contents

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.

To simplify the implementation of IPAC device drivers which shall 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.

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

TPCI100 (TEWS)	Carrier for 2 IndustryPack® modules	(PCI)
TPCI200 (TEWS)	Carrier for 4 IndustryPack® modules	(PCI)
TCP201 (TEWS)	Carrier for 4 IndustryPack® modules	(compactPCI)
TCP211 (TEWS)	Carrier for 2 IndustryPack® modules	(compactPCI)
TCP212 (TEWS)	Carrier for 2 IndustryPack® modules	(compactPCI)
TCP213 (TEWS)	Carrier for 2 IndustryPack® modules	(compactPCI)
TCP220 (TEWS)	Carrier for 4 IndustryPack® modules	(compactPCI)
TAMC100 (TEWS)	Carrier for 1 IndustryPack® modules	(AMC)
TAMC200 (TEWS)	Carrier for 3 IndustryPack® modules	(AMC)
TAMC220 (TEWS)	Carrier for 3 IndustryPack® modules	(AMC)
TPCE200 (TEWS)	Carrier for 4 IndustryPack® modules	(PCIexpress)

2 Installation

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

installer_32bit.exe	Installation tool for 32bit systems (Windows XP or later)
installer_64bit.exe	Installation tool for 64bit systems (Windows XP or later)
tewspcicarrieramd64.cat	TEWS PCI IPAC Carrier Driver CAT-File (64-bit)
tewspcicarrieri386.cat	TEWS PCI IPAC Carrier Driver CAT-File (32-bit)
tewspcicarrier.inf	Windows installation script
tewsipacslotamd64.cat	TEWS IPAC Slot Driver CAT-File (64-bit)
tewsipacsloti386.cat	TEWS IPAC Slot Driver CAT-File (32-bit)
tewsipacslot.inf	Windows installation script
genericipacamd64.cat	Generic IPAC Driver CAT-File (64-bit)
genericipaci386.cat	Generic IPAC Driver CAT-File (32-bit)
genericipac.inf	Windows installation script
genericipac.h	Application include File for Generic IPAC Driver
genericipacExa.c	Example Application for Generic IPAC Driver
i386\	Directory containing driver files for 32bit Windows versions
amd64\	Directory containing driver files for 64bit Windows versions
CARRIER-SW-65-2.3.0.pdf	This document in PDF-Format
Release.txt	Information about the Device Driver Release
ChangeLog.txt	Release history

2.1 Software Installation

2.1.1 Windows 7

This section describes how to install the IPAC-Carrier Device Driver software on a Windows 7 (32bit or 64bit) operating system.

Execute the matching installer application on the distribution media, installer_32bit on a 32-bit system, installer_64bit on a 64-bit system. The installer will install all required driver files using an installation wizard.

After the Installation all drivers on the distribution media are available on the target system and the carrier devices and the slot devices should start.

Device entries for the installed IPAC boards should be created and shown in the device manager.

If generic devices are enabled (special configuration of an IPAC slot, see also 2.4) the device will be started immediately restarting the IPAC slot.

2.1.2 Windows XP

This section describes how to install the IPAC-Carrier Device Driver software on a Windows XP (32bit) operating system.

Execute the matching installer application (installer_32bit) on the distribution media. The installer will install all required driver files using an installation wizard. (Do not install any driver using the “new Device found” requester).

After the Installation all drivers on the distribution media are available on the target system and the carrier devices and the slot devices should start.

If devices do not start automatically during installation, execute the installer application again!

Device entries for the installed IPAC boards should be created and shown in the device manager.

If generic devices are enabled (special configuration of an IPAC slot, see also 2.4) the device will appear in the device manager after restarting the IPAC slot. The devices will not start immediately and the installer must be executed once again. After that the generic devices should start and are available for use.

2.1.3 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.

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

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

(3) Insert the IPAC-Carrier driver media and select "***Disk Drive***" in the dialog box.
Click "***Next***" button to continue.

(4) Now the driver wizard should find a suitable device driver on the installation media.
Click "***Next***" button to continue.

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

(6) If the system requests a restart, the system should be restarted after all pending device driver installation requests have been handled.

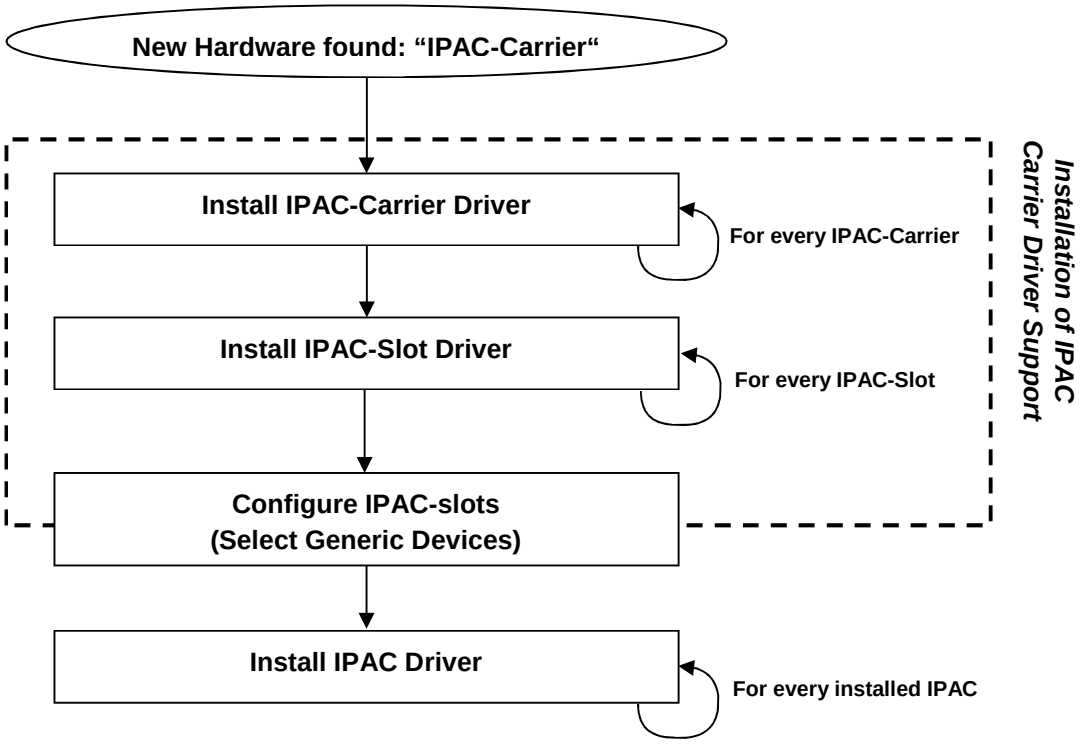
(7) Repeat the Instructions until drivers for all IPAC Carrier Boards are installed.

(8) Now “Generic IPAC” support can be configured for the IPAC slots. (see 2.4)

(9) The next step to do is to install the IPAC Slot Drivers. Simply repeat the above steps.

(10) Now copy all needed files to the desired target directories.

After successful installation the IPAC-Carrier device driver will start immediately



2.2 Confirming Driver Installation

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

1. Open the Windows Device Manager:

   a. For Windows 2000 / XP, open the "***Control Panel***" from "***My Computer***" and click the "***System***" icon and choose the "***Hardware***" tab, and then click the "***Device Manager***" button.

   b. For Windows 7, open the "***Control Panel***" from "***My Computer***" and then click the "***Device Manager***" entry.

2. Click the "**+**" in front of "***Embedded I/O***".
   The driver "**TPCI100 IPAC Carrier Board**" should appear for each installed TEWS PCI carrier board. ("TPCI100 …" should appear only for TCPI100 boards, the name should match the installed carrier board type).
   The driver "***TEWS IPAC Slot***" should appear for available IPAC slots.
   If generic device is selected for an IPAC slot with a mounted IPAC, the driver "**Generic IPAC (mounted to …)**" should appear for the device. Otherwise a device in the Device Manager (generally below of "Other Devices") should appear for each mounted IPAC.

2.3 Configure IPAC-Carrier Identification

TEWS IPAC Carrier driver supports naming installed IPAC Carrier devices. This may be a helpful feature to identify an IPAC special slot in the system, although an IPAC Carrier Board has been added or removed from the system since installation.

Once the board has been named, this name will identify IPAC Carrier devices of the same type at this specific slot (PCI-slot, compactPCI-slot etc.). If a board has been removed and is inserted later back to this location, the driver will use the configured naming again.

For setting the carrier name, just open the properties of the carrier device using the Windows Device Manager. Make a right-click to the IPAC Carrier Board device and click “Properties”. Change to the “Carrier Properties”-tab and select a “Carrier name”. After modifying the name and pressing “OK” the new configuration is stored and will be used after the next device restart.

To make a safe identification it is necessary that the assigned carrier names are unique for the system!

Remember that a device restart is necessary before the modifications are used!

Any IPAC carrier driver may request the slot location. The driver will receive a name for the IPAC slot, which is a concatenation of the specified “Carrier Name” and the name of the slot (e.g. “Slot A”). The IPAC drivers will generally support a function which reads out the identification string. The returned character string will look like “<Carrier Name> - Slot <n>”, e.g. “MyCarrier – Slot B”.

2.4 Configure Generic IPAC Device Support

It may be necessary to use IPACs which have no driver support matching to the TEWS carrier driver software (CARRIER-SW-65) or the available driver does not support the necessary functions. The carrier driver software offers a generic IPAC driver, which provides full access to the device resources of any IPAC.

If the generic IPAC driver should be used for a mounted IPAC, the IPAC slot must be configured to create a generic device for this slot. Therefore make a right-click to the TEWS IPAC Slot device and click “Properties”. Change to the “Properties of the IPAC Slot”-tab and select “Generic IPAC Slot Name”. After pressing “OK” the configuration is stored and will be used after the next device restart.

Remember that a device restart is necessary before the modifications are used!

After modification and restart the system may request a driver for the generic IPAC device. If the driver is not automatically started, the driver can be installed following the instructions of chapter 2.1.1., 2.1.2, or 2.1.3.

3 Generic IPAC Device Driver

The Generic IPAC Driver allows the use of IPAC boards where driver support is not available or does not match to the CARRIER-SW-65 support.

The generic IPAC driver is a kernel mode driver which allows the operation of IPAC devices on the Windows operating system.

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.

Because the generic IPAC device driver is stacked on the TEWS TECHNOLOGIES IPAC Carrier Driver, it is necessary to install also the appropriate IPAC carrier driver.

The generic IPAC device driver includes the following functions:

- configuration of IPAC slot parameters (clock-speed, bus-width)
- configuration of IPAC parameters (ID-, IO-, Memory space size and endianess)
- reading and writing of all IPAC spaces (ID-, IO- and Memory-Space)
- reads and writes with access-size of 8-, 16- and 32-bit
- special support of interrupt events (with and without vector)
- mapping of address spaces into the process context of the user application
- resetting IPAC slot
- reading the IPAC slot localization identifier (unique IPAC slot name)

If a device should be used as a generic device, it is necessary that the IPAC slot device is configured for generic device support. (See 2.4.)

3.1 Generic 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.

3.1.1 Opening a Generic IPAC Device

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

Pointer to a null-terminated string that specifies the name of the generic IPAC to open. The *lpFileName* string should be of the form **\\.\GenericIpac_*x*** to open the device *x*. The ending x is a one-based number. The first device found by the driver is \\.\GenericIpac_1, the second \\.\GenericIpac_2 and so on.

dwDesiredAccess

Specifies the type of access to the generic IPAC. For the 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 devices, 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 NULL for generic IPAC devices.

RETURN VALUE

If the function succeeds, the return value is an open handle to the specified generic 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(
     “\\\\.\\GenericIpac_1”,
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,              // no security attrs
     OPEN_EXISTING,     // generic 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()

3.1.2 Closing a Generic IPAC Device

The CloseHandle function closes an open generic IPAC device handle.

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

PARAMETERS

hDevice

Identifies an open generic IPAC device 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(…);

…

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

SEE ALSO

CreateFile(), Win32 documentation CloseHandle()

3.1.3 Generic 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 generic 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 *genericIpac.h*:

Value	Meaning
IOCTL_GENERICIPAC_CONFIGURE	Configure IPAC slot and map IPAC register spaces
IOCTL_GENERICIPAC_UNCONFIGURE	Remove IPAC register space mapping
IOCTL_GENERICIPAC_READ_U8	Read a buffer of 8-bit data from an IPAC space
IOCTL_GENERICIPAC_READ_U16	Read a buffer of 16-bit data from an IPAC space
IOCTL_GENERICIPAC_READ_U32	Read a buffer of 32-bit data from an IPAC space
IOCTL_GENERICIPAC_WRITE_U8	Write a buffer of 8-bit data to an IPAC space
IOCTL_GENERICIPAC_WRITE_U16	Write a buffer of 16-bit data to an IPAC space
IOCTL_GENERICIPAC_WRITE_U32	Write a buffer of 32-bit data to an IPAC space
… continued	… continued

Value	Meaning
IOCTL_GENERICIPAC_REGISTER_EVENT_VECTOR	Register an interrupt vector that shall be handled by the device
IOCTL_GENERICIPAC_WAIT_FOR_EVENT	Wait for an interrupt event
IOCTL_GENERICIPAC_ABORT_WAIT	Abort an ioctl call waiting for an interrupt event.
IOCTL_GENERICIPAC_MAP_SPACE	Map an IPAC space into the user's process. (Allow direct access)
IOCTL_GENERICIPAC_UMMAP_SPACE	Un-map an user mapped IPAC space
IOCTL_GENERICIPAC_RESET_DEVICE	Execute a an IPAC reset
IOCTL_GENERICIPAC_LOCALIZE_IPAC	Read the localization name of the IPAC slot

See behind for more detailed information on each control code.

To use these generic IPAC specific control codes the header file genericIpac.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()

3.1.3.1 IOCTL_GENERICIPAC_CONFIGURE

This function sets up the IPAC slot parameters IPAC clock speed and bus access size and specifies the size of the IPAC address spaces and maps them to the device. The function returns a base interrupt vector that is suggested to be used for the device.

The generic IPAC device must be configured before any other access to the device is allowed. Therefore this function should be called first before any of the other functions.

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

```
typedef struct
{
    BOOLEAN         clockSpeed;
    BOOLEAN         dataBusWidth;
    unsigned int    idSpaceSize;
    BOOLEAN         swapIdData;
    unsigned int    ioSpaceSize;
    BOOLEAN         swapIoData;
    unsigned int    memSpaceSize;
    BOOLEAN         swapMemData;
} GENERICIPAC_CONFIG_PAR;
```

clockSpeed

This parameter configures the IP clock rate. If 32MHz is configured, but not supported by the carrier board, 8MHz will be used. The following defines shall be used for configuration.

Define	Description
GENERICIPAC_8MHz	configure 8MHz IP-Clock-Speed
GENERICIPAC_32MHz	configure 32MHz IP-Clock-Speed

dataBusWidth

This parameter specifies the IP data bus width for memory space access. If the requested data bus width is not available, the supported bus width will be used. The following defines shall be used for bus width configuration.

Define	Description
GENERICIPAC_BUSWIDTH_8	configure IPAC slot for 8-bit data bus width
GENERICIPAC_BUSWIDTH_16	configure IPAC slot for 16-bit data bus width

idSpaceSize

This parameter specifies the size of ID address space supported on the IPAC. If this space shall not be used or is not supported, the size shall be set to 0.

swapIdData

This parameter specifies if the 16-bit and 32-bit access functions shall swap the data values of the ID-space. TRUE enables byte swapping, FALSE disables the swapping.
Generally the IPAC carrier driver tries to identify the endianess of the installed IPAC by checking the ID-Prom and tries to return data as big-endian values. If the IPAC uses mixed endianess between the spaces or the application needs little-endian data, this value can be used to correct the data endianess.

ioSpaceSize

This parameter specifies the size of IO address space supported on the IPAC. If this space shall not be used or is not supported, the size shall be set to 0.

swapIoData

This parameter specifies if the 16-bit and 32-bit access functions shall swap the data values of the IO-space. TRUE enables byte swapping, FALSE disables the swapping.
Generally the IPAC carrier driver tries to identify the endianess of the installed IP by checking the ID-Prom and tries to return data as big-endian values. If the IPAC uses mixed endianess between the spaces or the application needs little-endian data, this value can be used to correct the data endianess.

memSpaceSize

This parameter specifies the size of Memory space supported on the IPAC. If this space shall not be used or is not supported, the size shall be set to 0.

swapMemData

This parameter specifies if the 16-bit and 32-bit access functions shall swap the data values of the memory-space. TRUE enables byte swapping, FALSE disables the swapping.
Generally the IPAC carrier driver tries to identify the endianess of the installed IP by checking the ID-Prom and tries to return data as big-endian values. If the IPAC uses mixed endianess between the spaces or the application needs little-endian data, this value can be used to correct the data endianess.

The parameter *lpOutBuffer* must pass a pointer to an unsigned char buffer where the suggested basic interrupt vector (8-bit) will be stored. This interrupt vector may be used as the base value for the interrupt vectors used on the device. (This may be very helpful if the IPAC carrier board or driver can not recognize the IPAC slot generating the interrupt.)

EXAMPLE

```
#include “genericIpac.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
GENERICIPAC_CONFIG_PAR  confBuf;
UCHAR                   suggestedIntVector;

confBuf.clockSpeed      = GENERICIPAC_8MHz;         // 8MHz IP clock rate
confBuf.dataBusWidth    = GENERICIPAC_BUSWIDTH_16;  // Bus-width 16-bit
confBuf.idSpaceSize     = 0x40;       // 64 byte ID-space size
confBuf.swapIdData      = FALSE;      // no swapping (big-endian)
confBuf.ioSpaceSize     = 0x20;       // 32 byte ID register space
confBuf.swapIoData      = FALSE;      // no swapping (big-endian)
confBuf.memSpaceSize    = 0x100000;   // 1MB memory space
confBuf.swapMemData     = TRUE;       // swap data !!

…
```

```
…

/*
** configure and map IPAC Slot
*/
success = DeviceIoControl(   hDevice,
                             IOCTL_GENERICIPAC_CONFIGURE,
                             &confBuf,
                             sizeof(GENERICIPAC_CONFIG_PAR),
                             &suggestedIntVector,
                             sizeof(suggestedIntVector),
                             &NumBytes,
                             NULL);
if(success)
{
     printf(“Suggested interrupt vector: 0x%02x\n”, suggestedIntVector);
}
else
{
     ErrorHandler("Device I/O control error”);  // process error
}
```

ERROR CODES

Error Code	Description
ERROR_INSUFFICIENT_BUFFER	The input or output buffer is too small. Please check the parameter nOutBufferSize and nInBufferSize of the DeviceIoControl() function call.
ERROR_BUSY	The IPAC slot has already been configured.
ERROR_INVALID_ADDRESS	Mapping of IPAC slot space failed.
ERROR_INVALID_PARAMETER	Configuration of IPAC-slot failed.

All other returned error codes are system error conditions.

3.1.3.2 IOCTL_GENERICIPAC_UNCONFIGURE

This function unmaps the allocated IPAC spaces and allows a reconfiguration of the IPAC slot.

The parameters *lpInBuffer* and *lpOutBuffer* are not used and should be set to NULL.

Before calling this function all user-mapped spaces must be unmapped. And the device should be configured in passive state. (Disable interrupts on hardware).

EXAMPLE

```
#include “genericIpac.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;

/*
** unconfigure and unmap IPAC Slot
*/
success = DeviceIoControl(   hDevice,
                             IOCTL_GENERICIPAC_UNCONFIGURE,
                             NULL,
                             0,
                             NULL,
                             0,
                             &NumBytes,
                             NULL);
if(success)
{
     // success
}
else
{
     ErrorHandler("Device I/O control error”);  // process error
}
```

ERROR CODES

All returned error codes are system error conditions.

Error Code	Description
ERROR_DEVICE_IN_USE	There are active user-mapped spaces.

3.1.3.3 IOCTL_GENERICIPAC_READ_U8

This function reads a buffer of 8-bit data values from a specified address space of the generic IPAC device.

The parameter *lpInBuffer* must pass a pointer to the I/O-access buffer (*GENERICIPAC_IO_PAR*) to the device driver.

```
typedef struct
{
    unsigned int        ipacSpace;
    unsigned int        offset;
    unsigned int        numItems;
} GENERICIPAC_IO_PAR;
```

ipacSpace

This parameter specifies the IPAC address space that shall be accessed. The following address spaces are defined:

Define	Description
GENERICIPAC_IDSPACE	Access ID-space
GENERICIPAC_IOSPACE	Access IO-space
GENERICIPAC_MEMSPACE	Access memory-space

offset

This parameter specifies the address starting offset inside the specified address space. The offset is specified in byte.

numItems

This parameter specifies the number of 8-bit (unsigned char) values, which shall be read from the specified address space.

The parameter *lpOutBuffer* must pass a pointer to a buffer with a size of at least *numItems* 8-bit values. The read data will be returned in this buffer.

EXAMPLE

```
#include “genericIpac.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
GENERICIPAC_IO_PAR      ioBuf;
unsigned char           *pBuffer;
int                     bufSize;

…
```

```
…

// Read 8 byte from IO-space starting at address 0x10
ioBuf.ipacSpace    = GENERICIPAC_IOSPACE;
ioBuf.offset       = 0x10;            // Starting offset
ioBuf.numItems     = 8;

// Allocate input buffer
bufSize   = ioBuf.numItems * sizeof(unsigned char);
pBuffer   = (unsigned char*)malloc(bufSize);
if (pBuffer != NULL)
{
     // handle allocation error
}

success = DeviceIoControl(   hDevice,
                             IOCTL_GENERICIPAC_READ_U8,
                             &ioBuf,
                             sizeof(GENERICIPAC_IO_PAR),
                             pBuffer,
                             bufSize,
                             &NumBytes,
                             NULL);
if(success)
{
     printf(“IO-Data: 0x%02x  0x%02x … \n” , pBuffer[0] , pBuffer[1]);
}
else
{
     ErrorHandler("Device I/O control error”);  // process error
}
```

ERROR CODES

Error Code	Description
ERROR_INSUFFICIENT_BUFFER	The input or output buffer is too small. Please check the parameter nOutBufferSize and nInBufferSize of the DeviceIoControl() function call.
ERROR_INVALID_ADDRESS	Specified space has not been allocated.
ERROR_NOACCESS	Access exceeds limits of space.
ERROR_INVALID_PARAMETER	Invalid parameter specified.

All other returned error codes are system error conditions.

3.1.3.4 IOCTL_GENERICIPAC_READ_U16

This function reads a buffer of 16-bit data values from a specified address space of the generic IPAC device.

The parameter *lpInBuffer* must pass a pointer to the I/O-access buffer (*GENERICIPAC_IO_PAR*) to the device driver.

```
typedef struct
{
    unsigned int        ipacSpace;
    unsigned int        offset;
    unsigned int        numItems;
} GENERICIPAC_IO_PAR;
```

ipacSpace

This parameter specifies the IPAC address space that shall be accessed. The following address spaces are defined:

Define	Description
GENERICIPAC_IDSPACE	Access ID-space
GENERICIPAC_IOSPACE	Access IO-space
GENERICIPAC_MEMSPACE	Access memory-space

offset

This parameter specifies the address starting offset inside the specified address space. The offset is specified in byte.

numItems

This parameter specifies the number of 16-bit (unsigned short) values, which shall be read from the specified address space.

The parameter *lpOutBuffer* must pass a pointer to a buffer with a size of at least *numItems* 16-bit values. The read data will be returned in this buffer.

EXAMPLE

```
#include “genericIpac.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
GENERICIPAC_IO_PAR      ioBuf;
unsigned short          *pBuffer;
int                     bufSize;

…
```

```
…

// Read 16 16-bit words from ID-space starting at address 0x00
ioBuf.ipacSpace    = GENERICIPAC_IDSPACE;
ioBuf.offset       = 0x00;            // Starting offset
ioBuf.numItems     = 0x10;

// Allocate input buffer
bufSize   = ioBuf.numItems * sizeof(unsigned short);
pBuffer   = (unsigned short*)malloc(bufSize);
if (pBuffer != NULL)
{
     // handle allocation error
}

success = DeviceIoControl(   hDevice,
                             IOCTL_GENERICIPAC_READ_U16,
                             &ioBuf,
                             sizeof(GENERICIPAC_IO_PAR),
                             pBuffer,
                             bufSize,
                             &NumBytes,
                             NULL);
if(success)
{
     printf(“ID-Data: 0x%04x  0x%04x … \n” , pBuffer[0] , pBuffer[1]);
}
else
{
     ErrorHandler("Device I/O control error”);  // process error
}
```

ERROR CODES

Error Code	Description
ERROR_INSUFFICIENT_BUFFER	The input or output buffer is too small. Please check the parameter nOutBufferSize and nInBufferSize of the DeviceIoControl() function call.
ERROR_INVALID_ADDRESS	Specified space has not been allocated.
ERROR_NOACCESS	Access exceeds limits of space.
ERROR_INVALID_PARAMETER	Invalid parameter specified.

All other returned error codes are system error conditions.

3.1.3.5 IOCTL_GENERICIPAC_READ_U32

This function reads a buffer of 32-bit data values from a specified address space of the generic IPAC device.

The parameter *lpInBuffer* must pass a pointer to the I/O-access buffer (*GENERICIPAC_IO_PAR*) to the device driver.

```
typedef struct
{
    unsigned int        ipacSpace;
    unsigned int        offset;
    unsigned int        numItems;
} GENERICIPAC_IO_PAR;
```

ipacSpace

This parameter specifies the IPAC address space that shall be accessed. The following address spaces are defined:

Define	Description
GENERICIPAC_IDSPACE	Access ID-space
GENERICIPAC_IOSPACE	Access IO-space
GENERICIPAC_MEMSPACE	Access memory-space

offset

This parameter specifies the address starting offset inside the specified address space. The offset is specified in byte.

numItems

This parameter specifies the number of 32-bit (unsigned int) values, which shall be read from the specified address space.

The parameter *lpOutBuffer* must pass a pointer to a buffer with a size of at least *numItems* 32-bit values. The read data will be returned in this buffer.

EXAMPLE

```
#include “genericIpac.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
GENERICIPAC_IO_PAR      ioBuf;
unsigned int            *pBuffer;
int                     bufSize;

…
```

```
…

// Read 100 32-bit words from Memory-space starting at address 0x10000
ioBuf.ipacSpace    = GENERICIPAC_MEMSPACE;
ioBuf.offset       = 0x10000;         // Starting offset
ioBuf.numItems     = 100;

// Allocate input buffer
bufSize   = ioBuf.numItems * sizeof(unsigned int);
pBuffer   = (unsigned int*)malloc(bufSize);
if (pBuffer != NULL)
{
     // handle allocation error
}

success = DeviceIoControl(   hDevice,
                             IOCTL_GENERICIPAC_READ_U32,
                             &ioBuf,
                             sizeof(GENERICIPAC_IO_PAR),
                             pBuffer,
                             bufSize,
                             &NumBytes,
                             NULL);
if(success)
{
     printf("MEM-Data: 0x%08x  0x%08x … \n" , pBuffer[0] , pBuffer[1]);
}
else
{
     ErrorHandler("Device I/O control error");  // process error
}
```

ERROR CODES

Error Code	Description
ERROR_INSUFFICIENT_BUFFER	The input or output buffer is too small. Please check the parameter nOutBufferSize and nInBufferSize of the DeviceIoControl() function call.
ERROR_INVALID_ADDRESS	Specified space has not been allocated.
ERROR_NOACCESS	Access exceeds limits of space.
ERROR_INVALID_PARAMETER	Invalid parameter specified.

All other returned error codes are system error conditions.

3.1.3.6 IOCTL_GENERICIPAC_WRITE_U8

This function writes a buffer of 8-bit data values to the specified address space of the generic IPAC device.

The parameter *lpInBuffer* must pass a pointer to the I/O-access buffer (*GENERICIPAC_IO_PAR*) to the device driver.

```
typedef struct
{
    unsigned int        ipacSpace;
    unsigned int        offset;
    unsigned int        numItems;
} GENERICIPAC_IO_PAR;
```

ipacSpace

This parameter specifies the IPAC address space that shall be accessed. The following address spaces are defined:

Define	Description
GENERICIPAC_IDSPACE	Access ID-space
GENERICIPAC_IOSPACE	Access IO-space
GENERICIPAC_MEMSPACE	Access memory-space

offset

This parameter specifies the address starting offset inside the specified address space. The offset is specified in byte.

numItems

This parameter specifies the number of 8-bit (unsigned char) values, that shall be written to the specified address space.

The parameter *lpOutBuffer* must pass a pointer to a buffer containing the write-data. The buffer must have at least the length of *numItems* 8-bit values.

EXAMPLE

```
#include “genericIpac.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
GENERICIPAC_IO_PAR      ioBuf;
unsigned char           buffer[4] = {0x11, 0x22, 0x33, 0x44};
int                     bufSize;

…
```

```
…

// Write the data of the buffer to IO-space starting at address 0x16
ioBuf.ipacSpace    = GENERICIPAC_IOSPACE;
ioBuf.offset       = 0x16;            // Starting offset
ioBuf.numItems     = 4;

// Calculate size of buffer
bufSize   = ioBuf.numItems * sizeof(unsigned char);

success = DeviceIoControl(   hDevice,
                             IOCTL_GENERICIPAC_WRITE_U8,
                             &ioBuf,
                             sizeof(GENERICIPAC_IO_PAR),
                             buffer,
                             bufSize,
                             &NumBytes,
                             NULL);
if(success)
{
     // successful written
}
else
{
     ErrorHandler("Device I/O control error”);  // process error
}
```

ERROR CODES

Error Code	Description
ERROR_INSUFFICIENT_BUFFER	The input or output buffer is too small. Please check the parameter nOutBufferSize and nInBufferSize of the DeviceIoControl() function call.
ERROR_INVALID_ADDRESS	Specified space has not been allocated.
ERROR_NOACCESS	Access exceeds limits of space.
ERROR_INVALID_PARAMETER	Invalid parameter specified.

All other returned error codes are system error conditions.

3.1.3.7 IOCTL_GENERICIPAC_WRITE_U16

This function writes a buffer of 16-bit data values to the specified address space of the generic IPAC device.

The parameter *lpInBuffer* must pass a pointer to the I/O-access buffer (*GENERICIPAC_IO_PAR*) to the device driver.

```
typedef struct
{
    unsigned int        ipacSpace;
    unsigned int        offset;
    unsigned int        numItems;
} GENERICIPAC_IO_PAR;
```

ipacSpace

This parameter specifies the IPAC address space that shall be accessed. The following address spaces are defined:

Define	Description
GENERICIPAC_IDSPACE	Access ID-space
GENERICIPAC_IOSPACE	Access IO-space
GENERICIPAC_MEMSPACE	Access memory-space

offset

This parameter specifies the address starting offset inside the specified address space. The offset is specified in byte.

numItems

This parameter specifies the number of 16-bit (unsigned short) values, that shall be written to the specified address space.

The parameter *lpOutBuffer* must pass a pointer to a buffer containing the write-data. The buffer must have at least the length of *numItems* 16-bit values.

EXAMPLE

```
#include “genericIpac.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
GENERICIPAC_IO_PAR      ioBuf;
unsigned short          buffer[3] = {0x1111, 0x2222, 0x3333};
int                     bufSize;

…
```

```
…

// Write the data of the buffer to IO-space starting at address 0x8
ioBuf.ipacSpace    = GENERICIPAC_IOSPACE;
ioBuf.offset       = 0x8;             // Starting offset
ioBuf.numItems     = 3;

// Calculate size of buffer
bufSize   = ioBuf.numItems * sizeof(unsigned short);

success = DeviceIoControl(   hDevice,
                             IOCTL_GENERICIPAC_WRITE_U16,
                             &ioBuf,
                             sizeof(GENERICIPAC_IO_PAR),
                             buffer,
                             bufSize,
                             &NumBytes,
                             NULL);
if(success)
{
     // successful written
}
else
{
     ErrorHandler("Device I/O control error”);  // process error
}
```

ERROR CODES

Error Code	Description
ERROR_INSUFFICIENT_BUFFER	The input or output buffer is too small. Please check the parameter nOutBufferSize and nInBufferSize of the DeviceIoControl() function call.
ERROR_INVALID_ADDRESS	Specified space has not been allocated.
ERROR_NOACCESS	Access exceeds limits of space.
ERROR_INVALID_PARAMETER	Invalid parameter specified.

All other returned error codes are system error conditions.

3.1.3.8 IOCTL_GENERICIPAC_WRITE_U32

This function writes a buffer of 32-bit data values to the specified address space of the generic IPAC device.

The parameter *lpInBuffer* must pass a pointer to the I/O-access buffer (*GENERICIPAC_IO_PAR*) to the device driver.

```
typedef struct
{
    unsigned int        ipacSpace;
    unsigned int        offset;
    unsigned int        numItems;
} GENERICIPAC_IO_PAR;
```

ipacSpace

This parameter specifies the IPAC address space that shall be accessed. The following address spaces are defined:

Define	Description
GENERICIPAC_IDSPACE	Access ID-space
GENERICIPAC_IOSPACE	Access IO-space
GENERICIPAC_MEMSPACE	Access memory-space

offset

This parameter specifies the address starting offset inside the specified address space. The offset is specified in byte.

numItems

This parameter specifies the number of 32-bit (unsigned int) values, that shall be written to the specified address space.

The parameter *lpOutBuffer* must pass a pointer to a buffer containing the write-data. The buffer must have at least the length of *numItems* 32-bit values.

EXAMPLE

```
#include “genericIpac.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
GENERICIPAC_IO_PAR      ioBuf;
unsigned int            buffer[3] = {0x11111111, 0x22222222, 0x33333333};
int                     bufSize;

…
```

```
…

// Write the data of the buffer to Memory-space starting at address 0x1000
ioBuf.ipacSpace    = GENERICIPAC_MEMSPACE;
ioBuf.offset       = 0x1000;          // Starting offset
ioBuf.numItems     = 3;

// Calculate size of buffer
bufSize   = ioBuf.numItems * sizeof(unsigned int);

success = DeviceIoControl(   hDevice,
                             IOCTL_GENERICIPAC_WRITE_U32,
                             &ioBuf,
                             sizeof(GENERICIPAC_IO_PAR),
                             buffer,
                             bufSize,
                             &NumBytes,
                             NULL);
if(success)
{
     // successful written
}
else
{
     ErrorHandler("Device I/O control error”);  // process error
}
```

ERROR CODES

Error Code	Description
ERROR_INSUFFICIENT_BUFFER	The input or output buffer is too small. Please check the parameter nOutBufferSize and nInBufferSize of the DeviceIoControl() function call.
ERROR_INVALID_ADDRESS	Specified space has not been allocated.
ERROR_NOACCESS	Access exceeds limits of space.
ERROR_INVALID_PARAMETER	Invalid parameter specified.

All other returned error codes are system error conditions.

3.1.3.9 IOCTL_GENERICIPAC_REGISTER_EVENT_VECTOR

This function registers or unregisters an event vector to the interrupt handler of the specified generic IPAC device.

If a vector is registered, the driver will request an interrupt vector from the generic IPAC device, hence an IACK cycle will be executed to the IPAC. This may also be necessary if the IPAC clears the interrupt with an IACK cycle.

The specified vector must match the vector the IPAC will return with the IACK cycle, otherwise the event will not be announced and the interrupt may not be handled correctly.

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

```
typedef struct
{
    unsigned int        action;
    unsigned char       intVectorNo;
} GENERICIPAC_REGISTER_EVENT_VECTOR_PAR;
```

action

This parameter specifies the IPAC address space that shall be accessed. The following address spaces are defined:

Action	Description
GENERICIPAC_REGISTER	Register the specified vector to be handled for the IPAC
GENERICIPAC_UNREGISTER	Remove the specified vector from interrupt event handling

intVectorNo

This parameter specifies the interrupt vector (only 8-bit vectors are supported). Valid vector numbers are 1 up to 254. (0 is no valid vector number, 255 is used internally)

The parameter *lpOutBuffer* is unused and should be NULL.

EXAMPLE

```
#include “genericIpac.h”

HANDLE                                  hDevice;
BOOLEAN                                 success;
ULONG                                   NumBytes;
GENERICIPAC_REGISTER_EVENT_VECTOR_PAR vecBuf;

// Register vector 0xCC for interrupt event
vecBuf.action      = GENERICIPAC_REGISTER;
vecBuf.intVectorNo = 0xCC;
…
```

```
…

success = DeviceIoControl(   hDevice,
                             IOCTL_GENERICIPAC_REGISTER_EVENT_VECTOR,
                             &vecBuf,
                             sizeof(GENERICIPAC_REGISTER_EVENT_VECTOR_PAR),
                             NULL,
                             0,
                             &NumBytes,
                             NULL);
if(success)
{
     // vector successful registered
}
else
{
     ErrorHandler("Device I/O control error”);  // process error
}
```

ERROR CODES

Error Code	Description
ERROR_INSUFFICIENT_BUFFER	The input or output buffer is too small. Please check the parameter nOutBufferSize and nInBufferSize of the DeviceIoControl() function call.
ERROR_BUSY	For action GENERICIPAC_REGISTER: the vector has already been registered For action GENERICIPAC_UNREGISTER: the vector has not been registered
ERROR_INVALID_PARAMETER	Invalid parameter specified.

All other returned error codes are system error conditions.

3.1.3.10 IOCTL_GENERICIPAC_WAIT_FOR_EVENT

This function enables interrupts for the generic IPAC and waits for an interrupt event. The function will wait until an event occurs or the specified timeout has passed.

The interrupt normally have to be cleared by the application, e.g. by using the hardware access functions.

An interrupt event will be announced for registered interrupt vectors, if at least one vector is registered. If no vector has been registered, any interrupt on the specified slot will be detected.

The parameter *lpInBuffer* must pass a pointer to the wait parameter buffer (*GENERICIPAC_WAIT_EVENT_PAR*) to the device driver.

```
typedef struct
{
    unsigned int        flags;
    int                 timeout;
} GENERICIPAC_WAIT_EVENT_PAR;
```

flags

This parameter is an OR'ed value of flags specifying the wait job configuration. The following flags can be specified.

Flags	Description
GENERICIPAC_FL_ENA_INT0	Enable INT0 pin of the IPAC-slot and wait for interrupt.
GENERICIPAC_FL_ENA_INT1	Enable INT1 pin of the IPAC-slot and wait for interrupt.

timeout

This parameter specifies the timeout time in seconds. If -1 is specified, the function will wait without timeout.

The parameter *lpOutBuffer* must pass a pointer to an unsigned char buffer where the last received interrupt vector will be stored, if an event vector has been registered. Otherwise the value is not valid.

EXAMPLE

```
#include "genericIpac.h"

HANDLE                      hDevice;
BOOLEAN                     success;
ULONG                       NumBytes;
GENERICIPAC_WAIT_EVENT_PAR  evBuf;
unsigned char               retVector;

...
```

```
…

// register vectors (optional)
…

// Wait for interrupt event on INT0, timeout after 30 seconds
evBuf.flags =      GENERICIPAC_FL_ENA_INT0;     // Use INT0
evBuf.timeout =    30;                          // Timeout after 30 seconds

success = DeviceIoControl(   hDevice,
                             IOCTL_GENERICIPAC_WAIT_FOR_EVENT,
                             &evBuf,
                             sizeof(GENERICIPAC_WAIT_EVENT_PAR),
                             &retVector,
                             sizeof(retVector),
                             &NumBytes,
                             NULL);
if(success)
{
     printf(“received interrupt vector: 0x%02x\n”, retVector);
}
else
{
     ErrorHandler("Device I/O control error”);  // process error
}
```

ERROR CODES

Error Code	Description
ERROR_INSUFFICIENT_BUFFER	The input or output buffer is too small. Please check the parameter nOutBufferSize and nInBufferSize of the DeviceIoControl() function call.
ERROR_IO_DEVICE	Enabling interrupt failed.
ERROR_BUSY	There is already an active wait job for this generic IPAC device.
ERROR_SEM_TIMEOUT	The specified timeout has passed without an interrupt event.
ERROR_REQUEST_ABORTED	The wait job has been aborted.

All other returned error codes are system error conditions.

3.1.3.11 IOCTL_GENERICIPAC_ABORT_WAIT

This function aborts the interrupt event wait job on the specified generic IPAC device. The waiting function will return an error (ERROR_REQUEST_ABORTED).

The parameters *lpInBuffer* and *lpOutBuffer* are not used and should be set to NULL.

EXAMPLE

```
#include “genericIpac.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;

/*
** abort wait job
*/
success = DeviceIoControl(   hDevice,
                             IOCTL_GENERICIPAC_ABORT_WAIT,
                             NULL,
                             0,
                             NULL,
                             0,
                             &NumBytes,
                             NULL);
if(success)
{
     // success
}
else
{
     ErrorHandler("Device I/O control error”);  // process error
}
```

3.1.3.12 IOCTL_GENERICIPAC_MAP_SPACE

This function maps a specified IPAC address space into the application environment. The function will return a virtual address that can be used for direct access of IPAC registers and memory. The function will always map the complete size of the IPAC address space as it has been configured with IOCTL_GENERICIPAC_CONFIGURE.

Virtual addresses allow full access to the device, the application has to care about the limits of the IPAC spaces and endianess.

Mapped spaces should always be unmapped before leaving the application.

The parameter *lpInBuffer* must pass a pointer to an unsigned int value. The value specifies the IPAC space that shall be mapped. The following defines must be used for space selection.

Define	Description
GENERICIPAC_IDSPACE	Access ID-space
GENERICIPAC_IOSPACE	Access IO-space
GENERICIPAC_MEMSPACE	Access memory-space

The parameter *lpOutBuffer* must pass a pointer to a map buffer (*GENERICIPAC_MAP_PAR*) where mapping information will be returned.

```
typedef struct
{
        unsigned int            size;
        void                    *ptr;
} GENERICIPAC_MAP_PAR;
```

size

This value returns the size of the mapped space. This value may be used to check the virtual space limits. The offset is specified in byte.

ptr

The pointer is the mapped virtual address, which can be used for direct IPAC access.

EXAMPLE

```
#include “genericIpac.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
int                     spaceSelector;
GENERICIPAC_MAP_PAR     mapBuf;
unsigned short          *usIpacPtr;
…
```

```
…

// Map memory space of generic IPAC
spaceSelector = GENERICIPAC_MEMSPACE;

success = DeviceIoControl(   hDevice,
                             IOCTL_GENERICIPAC_MAP_SPACE,
                             &spaceSelector,
                             sizeof(spaceSelector),
                             &mapBuf,
                             sizeof(GENERICIPAC_MAP_PAR),
                             &NumBytes,
                             NULL);
if(success)
{
     usIpacPtr = (unsigned short*)mapBuf.ptr;
     printf(“Space-size: %d Byte\n”, mapBuf.size);
     printf(“Mem-Data (direct): 0x%04x 0x%04x\n”,
               usIpacPtr [0],
               usIpacPtr [1]);
}
else
{
     ErrorHandler("Device I/O control error”);  // process error
}

…

// Unmap IPAC space
```

ERROR CODES

Error Code	Description
ERROR_INSUFFICIENT_BUFFER	The input or output buffer is too small. Please check the parameter nOutBufferSize and nInBufferSize of the DeviceIoControl() function call.
ERROR_NOT_ENOUGH_MEMORY	The specified IPAC space cannot be mapped. Maybe the specified IPAC space has not been mapped with IOCTL_GENERICIPAC_CONFIGURE.

All other returned error codes are system error conditions.

3.1.3.13 IOCTL_GENERICIPAC_UNMAP_SPACE

This function unmaps an IPAC address space that has been previously mapped with IOCTL_GENERIC_MAP_SPACE.

The parameter *lpInBuffer* must pass a pointer to the mapped virtual address (void*). The address must match the address IOCTL_GENERIC_MAP_SPACE has returned as argument ptr in the structure GENERICIPAC_MAP_PAR.

The parameter *lpOutBuffer* is not used and should be NULL.

EXAMPLE

```
#include “genericIpac.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
void                    *virtIpacAddr;

// Map IPAC space
…;
virtIpacAddr = …;

…

// Unmap space of generic IPAC
success = DeviceIoControl(   hDevice,
                             IOCTL_GENERICIPAC_UNMAP_SPACE,
                             &virtIpacAddr,
                             sizeof(virtIpacAddr),
                             NULL,
                             0,
                             &NumBytes,
                             NULL);
if(success)
{
     // success, virtIpacAddr is no longer valid
}
else
{
     ErrorHandler("Device I/O control error”);  // process error
}
```

ERROR CODES

Error Code	Description
ERROR_INSUFFICIENT_BUFFER	The input or output buffer is too small. Please check the parameter nOutBufferSize and nInBufferSize of the DeviceIoControl() function call.
ERROR_INVALID_PARAMETER	Invalid parameter, specified virtual address is not valid

All other returned error codes are system error conditions.

3.1.3.14 IOCTL_GENERICIPAC_RESET_DEVICE

This function executes an IPAC reset on the specified generic IPAC device. This function will reset the IPAC slot and the mounted device will be reset.

ATTENTION: The function will wait for the reset execution. This time may be depending on the hardware some hundred milliseconds.

The parameters *lpInBuffer* and *lpOutBuffer* are not used and should be set to NULL.

EXAMPLE

```
#include “genericIpac.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;

/*
** Rest mounted generic IPAC
*/
success = DeviceIoControl(   hDevice,
                             IOCTL_GENERICIPAC_RESET,
                             NULL,
                             0,
                             NULL,
                             0,
                             &NumBytes,
                             NULL);
if(success)
{
     // success
}
else
{
     ErrorHandler("Device I/O control error”);  // process error
}
```

ERROR CODES

Error Code	Description
ERROR_NOT_SUPPORTED	Reset is not supported for the IPAC slot of the generic device.

All other returned error codes are system error conditions.

3.1.3.15 IOCTL_GENERICIPAC_LOCALIZE_IPAC

This function returns a character string which can be used for the IPAC localization. The returned string is a concatenation of the Carrier-Board name and the IPAC Slot Number. (See also 2.2)

The parameter *lpInBuffer* is not used and should be NULL.

The parameter *lpOutBuffer* must pass a pointer to a buffer where the localization string (char*) can be copied to. If the specified buffer is too small the function will just return a string of the specified size, so make sure that the supplied is sufficient.

EXAMPLE

```
#include “genericIpac.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
char                    locNameBuf[100];

// Get localization string
success = DeviceIoControl(   hDevice,
                             IOCTL_GENERICIPAC_LOCALIZE_IPAC,
                             NULL,
                             0,
                             &locNameBuf,
                             100,          // buffer length = 100 byte
                             &NumBytes,
                             NULL);
if(success)
{
     printf(“IPAC mounted to: ‘%s’ \n”, locNameBuf);
}
else
{
     ErrorHandler("Device I/O control error”);  // process error
}
```

ERROR CODES

Error Code	Description
ERROR_INSUFFICIENT_BUFFER	The input or output buffer is too small. Please check the parameter nOutBufferSize and nInBufferSize of the DeviceIoControl() function call.

All other returned error codes are system error conditions.

3.2 Programming Hints

This chapter describes necessary steps to use the features of generic IPAC devices.

3.2.1 Standard I/O Access

The generic IPAC device driver provides functions which allow access to the IPAC address spaces. These functions will always check if the access matches the specified space sizes. These functions will prevent application and system crashes due to invalid hardware accesses.

Before any access to the IPAC can be executed the generic device must be configured (by any process).

1. Configure generic device (IOCTL_GENERICIPAC_CONFIGURE)
   (Must be called once by any process)

2. Read / Write accesses can be called applications
   (IOCTL_GENERICIPAC_READ_Ux, IOCTL_GENERICIPAC_WRITE_Ux)
   (different processes may access the generic device, as long as the slot has been configured properly.)

3. Un-configure generic device (IOCTL_GENERICIPAC_UNCONFIGURE)
   (Must be called once by any process)

3.2.2 Direct I/O Access

The generic IPAC device driver allows direct access to the IPAC address spaces. This map function will just supply an address and the size of a space. The validation of accessed addresses is in the hand of the application. Therefor there is no overhead caused by the I/O-control functions.

Before any direct access to the IPAC can be executed the generic device must be configured (by any process) and the used space must be mapped for the application's process.

If the space is no longer used the space must be unmapped.

1. Configure generic device (IOCTL_GENERICIPAC_CONFIGURE)
   (Must be called once by any process)

2. Map IPAC address space(s)
   (IOCTL_GENERICIPAC_MAP_SPACE)
   (must be done by every process context that shall access)

3. Direct Read / Write accesses are possible
   (application process must have mapped the space)

4. Un-map IPAC address space(s)
   (IOCTL_GENERICIPAC_UNMAP_SPACE)
   (must be done for every process that has mapped a space)

5. Un-configure generic device (IOCTL_GENERICIPAC_UNCONFIGURE)
   (Must be called once by any process)

3.2.3 Interrupt Events

The generic IPAC device driver supports interrupt event handling. But this “interrupt handling” is limited and quite slow.

See below how interrupts can be used with the generic interrupt driver.

1. Configure generic device (IOCTL_GENERICIPAC_CONFIGURE)
   (Must be called once by any process)
2. Make default configuration of the used IPAC
3. Register interrupt vector(s) (optional)
   (IOCTL_GENERICIPAC_REGISTER_EVENT_VECTOR)
4. Wait for interrupt event(s)
   (IOCTL_GENERICIPAC_WAIT_FOR_INTEVENT)
5. Clear interrupt
   (application process must clear interrupt on IPAC, INTx must be released)
6. Un-register interrupt vector(s) (optional)
   (IOCTL_GENERICIPAC_REGISTER_EVENT_VECTOR)
7. Un-configure generic device (IOCTL_GENERICIPAC_UNCONFIGURE)
   (Must be called once by any process)

Because the wait function is limited and can just be used once a time, it may be useful to implement an “interrupt process” that handles interrupts for all process using the generic device. This “interrupt process” may just handle the steps 4 and 5 in a loop.

3.2.3.1 Interrupt (Event) Vectors

If no interrupt vector is registered, the generic driver will handle all interrupts that can be associated with the generic IPAC. Without registering an interrupt vector, no IACK cycle will be executed and there is no usable interrupt vector returned by the waiting function

If at least one interrupt vector is registered, the generic IPAC driver will execute an IACK cycle and will just handle interrupts of registered vectors.

Interrupt vectors may be useful if interrupts cannot be assigned to a unique IPAC slot (e.g. detection limited by used carrier board).

If an IPAC needs an IACK cycle to clear the interrupt, at least one vector must be registered.

3.2.3.2 Wait for Interrupt Handling Cycle

The following steps describe how interrupts are handled with the generic IPAC device driver.

(1)	*application*	Execute wait for interrupt event (blocking)
(2)	*generic driver (I/O-request)*	enable INT0 (and/or) INT1
(3)	*IPAC*	generates interrupt (INTx)
(4)	*generic driver (ISR)*	disable INT0 (and) INT1
(5)	*generic driver (DPC)*	get vector and unblock application
(6)	*application*	acknowledge interrupt on IPAC (release INTx)