WINCOR NIXDORF

HighPrint

HighPrint Driver Software

Version 1.2

Programming manual

Do you have ...

... any questions or suggestions regarding this manual?

Please note the order number or the date of issue of this manual and refer it to:

address:
WINCOR NIXDORF International GmbH
Handbuchredaktion BD PSD 5
33094 Paderborn
Germany

... any technical questions or problems?

Please refer to the WINCOR NIXDORF Customer Care GmbH.

E-mail: CCC.germany@wincor-nixdorf.com

HighPrint Driver Software

Version 1.2

Edition January 2008

© Wincor Nixdorf International GmbH 2008

All rights, including rights of translation and rights of reproduction by reprinting, copying or similar methods, even of parts, are reserved.
Any violations give rise to a claim for damages.

All rights, including rights created by patent grants or registration of a utility model or design, are reserved. Delivery subject to availability; right of technical modifications reserved.

All names of hardware and software products mentioned in this manual are trade names and/or trademarks of their respective manufacturers.

Contents

Introduction

The **HighPrint Driver Software** supports printers of type **HighPrint 4905** and **HighPrint 4915** with serial, parallel and USB interface. HighPrint 4915 is here defined as printer family consisting of HighPrint 4915, HighPrint 4915xe, and HighPrint 4915xa.

The driver software can be used with **Windows Vista** (32 and 64 bit), **Windows XP**, and **Windows 2000**.

The driver software combines two different ways of printing. On the one hand it is possible to print via the standard Windows printer interface provided by the GDI. In particular it is possible to use standard applications like Word, Acrobat Reader, and Internet Explorer. So the corresponding file formats (doc, pdf, html) can be printed without any problems. For this the HighPrint Driver Software includes all modules required by the Windows spool system: A GDI driver, a so-called language monitor for bi-directional communication, and an INF file for installation.

On the other hand it is possible to develop applications that directly talk to the printer. This is in general necessary for special applications that have to use functions of the printer that are not supported by the Windows GDI. An example is a passbook application which handles the passbook's magnetic stripe.

For such applications a special device handler called HPRDH.DLL is provided by the HighPrint Driver Software. Its interface is described in this document. By using this interface an application gains complete control of the printer.

A description of the GDI interface is not necessary here. For this we refer you to the Windows documentation.

So in summary the HighPrint Driver Software contains

- standard driver(s) for Windows and
- a device handler for special applications.

Installation

The HighPrint Driver Software can be installed on Windows Vista, Windows XP, and Windows 2000. This is possible using one of various installation procedures, which are described in the following chapters. All these installation procedures require you to have administrator rights.

Installation with HighPrint 4915 CD

For driver installation it is recommended to use the product CD supplied with the printer. This process will install the complete HighPrint driver software including device handler.

For USB devices we also recommend to run the installation before connecting the printer.

Start by inserting the product CD. Should the user interface fail to start automatically, run the HighPrint4915 executable file in the CD's root directory.

Click on "*Install printer driver*", then follow the installation program prompts.

If you want to run the HighPrint on a COM or LPT channel, then the installation ends with the setup of an appropriate printer object in the printer folder. This last step is omitted if the device is to be run on USB. In this case the printer object is automatically set up by Windows as soon as the printer is connected.

Unattended Installation

This chapter is aimed at system integrators, administrators and other IT professionals whose task is to install the HighPrint driver software on one or more systems. It is often helpful to be able to run an installation program that can execute completely unattended.

For exactly this situation you will find a program named Setup.exe on the HighPrint CD. The program takes the settings needed for the installation procedure from a control file in (text-based) INI format. This is where the settings are to be defined that otherwise, in attended mode, the user selects in a dialog with the installation program.

Please adapt this control file to your installation procedure and then pass it to the program with the appropriate call parameter. At this point it is also possible to specify a log file which instructs the program to log the installation procedure.

If the printer is to be run on a COM or LPT channel, the installation ends by setting up an appropriate printer object in the printer folder. This last step is omitted if the device is to be run on USB. In this case the printer object is automatically set up by Windows as soon as the printer is connected.

Syntax: `setup.exe <inifile> [<logfile>]`

`<inifile>` Path and name of the INI file. If the path of the ini file is not indicated, the path of the application file (setup.exe) is used.

`<logfile>` Path and name of the log file (optional; if not specified no log file is written). If the path of the log file is not specified then the path of the INI file parameter is used.

Examples:

```
setup.exe highprint.ini
```

```
setup.exe "c:\program\highprint.ini" "c:\program\logs\highprint.log"
```

Please note: For installing the printer driver software you need **administration rights** on your machine.

Steps of the Unattended Setup

The installation process depends on the INI file and the command-line parameters but can be summarized as follows:

- Running setup.exe
- Create log file
- Process INI file:
  - Identify source directory for printer files and name of INF file
  - Copy DEVICE_HANDLER_FILES (e.g. HPRDH.dll) from source to destination directory
  - Copy DRIVER_FILES (e.g. Hprw2k.inf, Hprw2k.cat) from source directory to destination directory
  - Set INF file for installation process to the local copy (target directory)
- Driver installation process using the specified INF file
  (COM/LPT: Setup adds the printer to the "Printers and faxes folder"
  USB: Setup prepares the driver for plug&play mechanism)
- Show installation status (e.g. Printer driver successfully installed)
- End of installation

For details see the INI and LOG files.

Exit Code

If the installation was successfully completed you will get the message "Printer driver successfully installed." from the log file and an exit code (also named error code) equal to 0.

If the setup program exits prematurely you will receive an exit code different from 0. The setup also returns an error message that describes the failing operation. The exit codes correspond to the system error codes from the Microsoft operating system but the detailed error messages are application-specific.

Please note: If the installation fails the target files will not be deleted and registry keys will not be removed or reset.

For a complete list of error codes provided by the operating system, see System Error Codes from the Microsoft website.

Log File

A log file typically consists of a list of events in chronological order. This file can be opened using any text editor (e.g. Notepad).

In order to verify that the printer has been installed properly have a look at the last line but one.

If this line contains "**Printer driver successfully installed.**" the installation has completed without errors.

The log file will indicate "**Installation not completed. Printer driver not installed.**" after any error during the installation process. In this case you need to find the problem and correct it. It requires no detective work to find the problem, simply go back line by line and evaluate the messages starting with "ERROR:"

Example:

Log-file	Description
`[***Start unattended printer driver installation 13.07.2005 16:29:17***]`	Start tag with date/time information. The following entry only contains a timestamp.
...	**...**
`[16:29:17] Processing INI-file D:\uasetup\param2.ini`	The operation that caused the error.
`[16:29:17] ERROR: Bad INI-file. Parameter INF_FILE not found or empty.`	Detailed description from setup.exe of the error.
`[16:29:17] ERROR: Program exit code 87 (HEX 00000057): One of the parameters was invalid.`	Exit code and operating system information for this exit code.
`[16:29:17] Installation not completed. Printer driver not installed.`	This line indicates a problem.
`[16:29:17] End of installation.`	End tag with timestamp.

INI File

The INI file contains the following sections and parameters:

[GENERAL]	Section with general settings
INF_FILE	Path and name of the inf file. All driver files have to be in this directory. If no directory is indicated, the path of the INI file is used or (if missing) the path of the exe file.
PRINTER_DRIVER_MODEL_NAME	Name of the printer driver to be installed. This name is assigned by the printer driver manufacturer and may be different from the product label.
DRIVER_FILES_TARGET_FOLDER	Target folder for the driver files. You may also use environment variables to specify the target folder. As usual, these must be put between % characters. The driver files to be copied must be listed in section [DRIVER_FILES] (see below). If this key is missing in the INI file or if no folder was specified to the right of the equal sign (=), no driver files will be copied. **Example:** %ProgramFiles%\Wincor Nixdorf\HighPrint
DEVICE_HANDLER_TARGET_FOLDER	Target folder for the device handler files. You may also use environment variables to specify the target folder. As usual, these must be put between % characters. The driver files to be copied must be listed in section [DEVICE_HANDLER_FILES] (see below). If this key is missing in the INI file or if no folder was specified to the right of the equal sign (=), no driver files will be copied. Please select a folder that is part of the relevant search pathes. **Example:** %systemroot%\system32

PORT	Port name: COM1:, ..., COM255:, LPT1:, ..., LPT4:, USB: The colon (:) behind the port name is necessary. For COM ports you have to set the transmission parameters in a separate section named exactly like the Port (see below). Please ensure that the name of that section exactly matches the port name (e.g. [COM1:].
[COM<x>:]	Settings for serial interface
BAUD	2400, 4800, 9600, 19200
PARITY	NONE, EVEN, ODD
DATABITS	7, 8
STOPBITS	1, 2
PROTOCOL	DTR, XON/XOFF
[DRIVER_FILES]	List of driver files
HPRW2K.INF HPRW2K.CAT HPR4915.GPD HPR4905.GPD WNHPR.DLL WNHPRLM.DLL HPRDH.DLL	This is the list of all files that are necessary to run one of the supported printers in 32-bit Windows versions.
HPRW64.INF HPRW64.CAT HPR4915.GPD HPR4905.GPD WNHPR.DLL WNHPRLM.DLL HPRDH.DLL	This is the list of all files that are necessary to run one of the supported printers in Windows Vista (64 bit). Please note that there are 64 bit versions of WNHPR.DLL and WNHPRLM.DLL. The device handler HPRDH.DLL is only available as 32 bit DLL for use in 32 bit applications.
[DEVICE_HANDLER_FILES]	List of device handler files
HPRDH.DLL	The device handler consists only of one single file.

Example for USB Port

```
[GENERAL]
INF_FILE=F:\Driver\Hprw2k.inf
PRINTER_DRIVER_MODEL_NAME=Wincor Nixdorf HighPrint 4915
DRIVER_FILES_TARGET_FOLDER=%ProgramFiles%\Wincor Nixdorf\HighPrint
DEVICE_HANDLER_TARGET_FOLDER=%SystemRoot%\system32
PORT=USB:

[DRIVER_FILES]
HPRW2K.INF
HPRW2K.CAT
HPR4915.GPD
HPR4905.GPD
WNHPR.DLL
WNHPRLM.DLL
HPRDH.DLL

[DEVICE_HANDLER_FILES]
HPRDH.DLL
```

Example for Serial Port COM2

```
[GENERAL]
INF_FILE=A:\Hprw2k.inf
PRINTER_DRIVER_MODEL_NAME=Wincor Nixdorf HighPrint 4915
DRIVER_FILES_TARGET_FOLDER=%ProgramFiles%\Wincor Nixdorf\HighPrint
DEVICE_HANDLER_TARGET_FOLDER=%SystemRoot%\system32
PORT=COM2:

[COM2:]
BAUD=19200
PARITY=EVEN
DATABITS=8
STOPBITS=1
PROTOCOL=DTR

[DRIVER_FILES]
HPRW2K.INF
HPRW2K.CAT
HPR4915.GPD
HPR4905.GPD
WNHPR.DLL
WNHPRLM.DLL
HPRDH.DLL

[DEVICE_HANDLER_FILES]
HPRDH.DLL
```

Example for Parallel Port LPT1

```
[GENERAL]
INF_FILE=A:\Hprw2k.inf
PRINTER_DRIVER_MODEL_NAME=Wincor Nixdorf HighPrint 4915
DRIVER_FILES_TARGET_FOLDER=%ProgramFiles%\Wincor Nixdorf\HighPrint
DEVICE_HANDLER_TARGET_FOLDER=%SystemRoot%\system32
PORT=LPT1:

[DRIVER_FILES]
HPRW2K.INF
HPRW2K.CAT
HPR4915.GPD
HPR4905.GPD
WNHPR.DLL
WNHPRLM.DLL
HPRDH.DLL

[DEVICE_HANDLER_FILES]
HPRDH.DLL
```

Installation with Add Printer Wizard

Wizards are a type of property sheet that provide a simple and powerful way to guide users through complex procedures. For printer driver installations Windows provides the Add Printer Wizard.

How the installation is to be started depends on the physical interface of the printer. For HighPrint printers with a serial or parallel interface the driver is to be installed from the Printers folder by starting the Add Printer Wizard explicitly. For printers with a USB interface the Add Printer Wizard is started automatically when they are connected.

In any case follow the instructions of the installation wizard.

If you have a serial printer please ensure that the configuration of the serial port matches the printer's settings. It is recommended to use HW flow control and a high baud rate (e.g. 19200).

Please note that this type of installation installs only the Windows printer driver. If the device handler HPRDH.DLL is also required, it has to be copied separately. Please make sure that you select a target directory where the applications will be able to find the DLL, (e.g. %SystemRoot%\system32).

Deinstallation

This chapter describes a procedure you can use to remove the HighPrint Driver Software from your system. It consists of two steps:

1. Remove all printer objects associated with the printer driver:

   a) Open the **Printers** folder.

   b) For all printer objects associated with the HighPrint driver right-click the printer object and click **Delete**.

2. Remove the printer driver:

   Windows Vista

   a) Open the **Printers and Faxes** folder.

   b) On the **File** menu, click **Run as Administrator** and select **Server Properties**.

   c) On the **Drivers** tab, click the HighPrint driver, and then click **Remove**.

   Windows XP

   a) Open the **Printers and Faxes** folder.

   b) On the **File** menu, click **Server Properties**.

   c) On the **Drivers** tab, click the HighPrint driver, and then click **Remove**.

   Windows 2000

   Windows 2000 has no user interface feature to delete printer drivers. So you have to manually remove the driver.

   For this follow the steps described in article "Steps to Manually Remove and Reinstall a Printer Driver" from the Microsoft Knowledge Base (article ID 135406).

Configuration

There are a few registry parameters which affect the behavior of the driver software. The parameters reside in the registry root of the language monitor. It is:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Print\Monitors\HighPrint Monitor

Parameter	Type	Description
Wait4PaperTime	REG_DWORD	Parameter that affects print jobs with paper source *Manual Feed*, *Manual Feed User 1*, or *Manual Feed User 2.* (See chapter Paper Source.) It specifies the time in milliseconds how long the driver is to wait for paper at the beginning of a GDI print job before it returns to the spooler, enabling the user to cancel the job. The value 0 has a special meaning. In this case the print data is immediately sent to the printer, whether or not paper is available. Default : 0 Note: This parameter must be set to 0 if the radio button "Print directly to the printer" is active. Open the printer folder and select the printer properties to see this radio button.
TraceLevel	REG_DWORD	Trace level for device handler HPRDH (see chapter *Trace and Logging* for details)
LmTraceLevel	REG_DWORD	Trace level for language monitor (see chapter *Trace and Logging* for details)

Please note that in general the spooler needs to be restarted for changes to take effect.

Driver Specifics

This chapter describes the options of the GDI printer driver which are specific to the printer. For standard options and settings we refer you to the Windows documentation.

Paper Forms

Standard Forms

The following standard forms are supported by the driver:

A4	210 x 297 mm
A5	148 x 210 mm
Letter	8 1/2 x 11
Statement	5 1/2 x 8 ½
Executive	7 1/4 x 10 ½
B5 (JIS)	182 x 257 mm
Quarto	2 15 x 275 mm
Envelope #9	3 7/8 x 8 7/8 in
Envelope #10	4 1/8 x 9 1/2 in
Envelope #11	4 1/2 x 10 3/8 in
Envelope #12	4 3/4 x 11 in
Envelope #14	5 x 11 1/2 in
Envelope DL	110 x 220 mm
Envelope C5	162 x 229 mm
Envelope C6	114 x 162 mm
Envelope C65	114 x 229 mm
Envelope B5	176 x 250 mm
Envelope B6	176 x 125 mm
Envelope	110 x 230 mm
Envelope Monarch	3 7/8 x 7 1/2 in
6 3/4 Envelope	3 5/8 x 6 1/2 in
Japanese Postcard	100 x 148 mm
9 x 11 in	9 x 11 in
Envelope Invite	220 x 220 mm
A5 Extra	174 x 235 mm
B5 (ISO) Extra	201 x 276 mm

Driver-Defined Forms

The following driver-defined forms are supported:

A5 Transverse	210 x 148 mm
A6	105 x 148 mm
A6 Transverse	148 x 105 mm
Eurocheque	3 2/6 x 5 9/10 in
EZUE	4 1/6 x 5 9/10 in

User-Defined Forms

In addition to using pre-defined formats, users may also define new formats. The following table specifies the maximum and minimum sizes permitted by the printer:

Width: 7 - 24 cm (2.76 - 9.45 inches)
Height: 7 - 50 cm (2.76 - 19.68 inches)

In order to define your own formats open the Printers folder and select Server Properties in the File menu.

Printable Area

Like with most matrix printers, form printing is subject to certain restrictions. Thus, the HighPrint family of printers is unable to print on the margins. If print elements such as texts or graphical elements are placed in these margin areas, the print system will mask them out.

The non-printable margins are defined as follows:

Top margin	2.7 mm
Bottom margin	3.5 mm
Right margin	2.54 mm
Left margin	2.54 mm

Paper Source

Driver Options

The driver supports the following paper feed options:

	4905	4915
Manual Feed	X	X
Manual Feed User 1	x	X
Manual Feed User 2	x	X
Sheet Feeder	–	X

If one of the first three options is selected, users are expected to manually feed paper to the printer. The difference between these three options consists in the handling of the user LEDs, which are part of the printer operating panel. Print jobs for which “Manual Feed” is selected cause none of the two LEDs to light up, while with "Manual Feed User x" the appropriate user LED lights up.

As these three feed modes are available it is possible to identify the print jobs from different workstations; therefore, the correct form can then be manually inserted into the printer.

If the "Sheet Feeder" is selected, the paper is automatically pulled from the optional feeder; no LED will light up. Note that a sheet feeder is available only for the 4915 and its successor products.

Automatically Select

In addition to the above paper feed modes. Windows supports the option to select “Automatically Select” for the paper hopper. In this case, the selection of the paper feed is carried out based on the selected paper size and the link between paper paths and paper formats made in the “Device Settings“ under “Forms to Tray Assignment“.

Note: If "Automatically Select" is selected for a print job, no dialog window will appear at the beginning of the job, independent of the setting of configuration parameter Wait4PaperDialog.

Fonts

As the printer is graphics-enabled, it is in principle possible to print all Windows fonts. In addition, it is possible to use the GDI to select and use the character sets defined in the printer. This may accelerate printing in particular for printers with serial interfaces.

Some particular characteristics of the printer-specific fonts are described in the following:

Device Fonts

The following printer fonts are supported by the driver:

Roman 5cpi	10 pt
Roman 6cpi	10 pt
Roman 10cpi	10 pt
Roman 12cpi	10 pt
Roman 15cpi	7 pt
Roman 17cpi	10 pt
Roman 20cpi	10 pt
Roman PS	10 pt
Sans Serif 5cpi	10 pt
Sans Serif 6cpi	10 pt
Sans Serif 10cpi	10 pt
Sans Serif 12cpi	10 pt
Sans Serif 15cpi	7 pt
Sans Serif 17cpi	10 pt
Sans Serif 20cpi	10 pt
Sans Serif PS	10 pt
OCR-A 10cpi	10 pt
OCR-B 10cpi	10 pt

Note: Font *OCR-A 10cpi* is defined only in part; the consequence being that e.g. the character 169 is printed as a question mark. See the HighPrint 4915 Programming Guide, where the precise character assignment is described.

Notes on Code Pages

Note the following concerning the use of the printer fonts in a GDI print job:

To achieve an optimal printing quality with the HighPrint 4915 make sure that the character generator corresponding to the ANSI code page of the Windows system used (e.g. 1252) has been loaded into the printer. See also the CD that is shipped with the printer.

The printer usually uses a character generator defined on the basis of code page 437. If this character generator is active in the printer, the printing of the characters higher than code 127 is under Windows normally not printed as expected.

For example: Character 169 (copyright character © in Windows codepage 1252) is printed as a semi graphic character with a loaded 437-type character generator.

These comments do not apply to characters 32 to 126, where all code pages are identical.

Paper Quality

This parameter exists only for the HighPrint 4905. It is used to activate the so-called passbook processing.

Two options are available: "Standard" and "Thicker than 0.15 mm".

Make sure to use always the second option for paper stock that is thicker than 0.15 mm (e.g. passbooks and multi-copy forms).

All envelope forms cause the driver to automatically activate the passbook mode.

Device Handler Specifics

The device handler HPRDH.DLL provides an interface which enables applications to use the full functionality of the printer. For details see the SDK.

In the following you will find a few remarks regarding the use of the device handler.

- **Locally connected printers are supported**

The device handler can only establish a connection to a locally connected printer. If you need to access the printer device handler over the network or by more than one application, you have to write a dedicated server process.

- **Bi-directional support must be activated**

In order to communicate with the printer over the device handler you should make sure that bi-directional support is activated. See the Ports register card in the driver settings. After the installation the bi-directional support is active by default.

- **Synchronization with print queue**

The process that has a connection to the printer by means of the device handler shares the device with the spooler process. Access to the device is synchronized with the spooler's print queue by means of the functions PrtClaim() and PrtRelease(). Further synchronization is not necessary.

Software Development Kit

A Software Development Kit is available for the development of an application that uses the special device handler HPRDH.DLL (32 bit). It includes

– HPRDH.H, a C-style header file, and

– HPRDH.LIB, the import library required for linking.

In addition, the source code of a running sample application is included.

All definitions and examples use the C programming language.

Software Interface

The following functions facilitate communication between the application and the dynamic-link library HPRDH.DLL:

PrtOpen	opens the printer
PrtClose	closes the printer
PrtClaim	claims the printer for exclusive access
PrtRelease	releases the printer
PrtWrite	writes data to the printer
PrtRead	reads information from printer

Please note that all functions follow the C calling convention. The DLL is a 32 bit binary.

To ensure that the device handler can also be used in programming environments that do not support the C calling convention (such as Visual Basic), a compatible list of functions following the Pascal calling convention is available. The names of these functions correspond to those mentioned above, but with a leading *p*. For example: *pPrtWrite*, *pPrtRead*, etc.

Note concerning *PrtOpen*: *PrtOpen* is strictly speaking not a function, but a macro, which accesses with a defined UNICODE constant the function *PrtOpenW*, otherwise the function *PrtOpenA*. Similarly, there are the functions *pPrtOpenW* and *pPrtOpenA*, a function *pPrtOpen* does not exist.

See also the chapter *Using the Device Handler from Visual Basic*.

PrtOpen

Synopsis

DWORD WINAPI PrtOpen (HANDLE *phPrt, LPTSTR lpPrinterName)

Description

The *PrtOpen* function retrieves a handle to the specified printer. The same printer cannot be opened by another application at the same time.

Parameters

phPrt
Pointer to a variable that receives a handle to the open printer.

lpPrinterName
Pointer to a null-terminated string that specifies the name of the printer as it is known in the system (for instance `"Wincor Nixdorf HighPrint 4915"`).

Example

```
DWORD   dwRet;              // return value from function
HANDLE  hPrt;               // handle to our printer object
TCHAR   tcPrinterName[] = TEXT( "Wincor Nixdorf HighPrint 4915");
LPTSTR  lpPrinterName = tcPrinterName;

// printer open call
dwRet = PrtOpen(&hPrt, lpPrinterName);

if (dwRet != PRT_NO_ERROR)
{
    ErrorProcedure(...);
}
```

PrtClose

Synopsis

DWORD WINAPI PrtClose (HANDLE *phPrt)

Description

The *PrtClose* function terminates the link to the printer. The handle is no longer valid.

Parameters

hPrt
Pointer to the printer handle returned by *PrtOpen*.

Example

```
DWORD   dwRet;                  // return value from function

dwRet = PrtClose(&hPrt);            // printer close call
if (dwRet != PRT_NO_ERROR)
{
    ErrorProcedure(...);
}
```

PrtClaim

Synopsis

DWORD WINAPI PrtClaim (HANDLE hPrt, DWORD dwTimeout)

Description

Claims the device for exclusive access. Claiming is necessary to synchronize the application with print jobs from the spooler. As long as the device is claimed all subsequent print jobs are queued but not processed.

Parameters

hPrt
Handle to the printer returned by *PrtOpen*.

dwTimeout
Maximum waiting time in milliseconds. The following defines have special meanings:

PRT_IMMEDIATELY	return immediately
PRT_INFINITE	wait until device is claimed

Example

```
DWORD   dwRet;       // return value from function
DWORD    dwTimeout;

dwTimeout = 10000;

// claim the device for exclusive access
dwRet = PrtClaim(hPrt, dwTimeout);
if (dwRet != PRT_NO_ERROR)
{
    if (dwRet == WAIT_TIMEOUT)
    {
        // printer is currently busy;
        // handle timeout according to your needs

        ClaimTimeoutHandler(...);

        // perhaps you want to display a corresponding
        // user dialog with retry and abort option
    }
    else
    {
        ErrorProcedure(...);
    }
}
```

PrtRelease

Synopsis

DWORD WINAPI PrtRelease (HANDLE hPrt)

Description

Releases the device so that the spooler's print queue is no longer blocked.

Parameters

hPrt
Handle to the printer returned by *PrtOpen*.

Example

```
DWORD   dwRet;              // return value from function

dwRet = PrtRelease(hPrt);   // release the device
if (dwRet != PRT_NO_ERROR)
{
    ErrorProcedure(...);
}
```

PrtWrite

Synopsis

DWORD WINAPI PrtWrite (HANDLE hPrt, LPBYTE pbWriteBuffer,
DWORD nNumberOfBytesToWrite,
LPDWORD lpNumberOfBytesWritten)

Description

The *PrtWrite* function writes *nNumberOfBytesToWrite* bytes of the contents of *pbWriteBuffer* to the printer. The function can be used to send data of any kind, normal print data as well as printer control sequences.

Please note that *PrtWrite* provides no character code mapping. So the printout depends on the loaded character generator.

If a control sequence requests a message from the printer then you will obtain the response with the *PrtRead* function. For a description of the available control sequences we refer you to the HighPrint Programming Guide.

Please note that the successful return does not indicate that all bytes are written to the printer. In general the function does not return an error if the buffer is partly transferred. Therefore you have to check the returned number of bytes written (see the example code below).

Parameters

hPrt
Handle to the printer returned by *PrtOpen*.
pbWriteBuffer
Pointer to the buffer containing the data to be written to the printer
nNumberOfBytesToWrite
Specifies the number of bytes to write to the printer
lpNumberOfBytesWritten
Pointer to the variable that receives the number of bytes written

Example

```
BYTE   bTextToWrite [] = "Text to be written to the printer";
DWORD  dwNoOfBytesWritten;
DWORD  nNumberOfBytesToWrite;
PBYTE  pBuffer;
BOOL   fAbort;
```

```
fAbort = FALSE;
// initialize our write parameters
nNumberOfBytesToWrite = strlen(bTextToWrite);
pBuffer = bTextToWrite;

while (nNumberOfBytesToWrite > 0 && fAbort == FALSE)
{
    dwNoOfBytesWritten = 0;

    dwRet = PrtWrite(hPrt, pBuffer, nNumberOfBytesToWrite,
                  &dwNoOfBytesWritten);

    if (dwRet == PRT_NO_ERROR)
    {
        AbortErrorProcedure(&fAbort, ...);
    }

    // all data written ?
    if (dwNoOfBytesWritten != nNumberOfBytesToWrite)
    {
        // check the printer's message queue
        if (isPrinterOkay(...))
        {
            // no error condition, so carry on
            // with sending data to the printer.
        }
        else
        {
            // printer has a problem
            ErrorHandler(...);
            break;
        }
    }

    if (dwRet != PRT_NO_ERROR)
    {
        // PrtWrite returned an error; it may be reasonable to
        // implement a similar error handling as the spooler:
        // retry the write for a certain time; simultaneously
        // display a dialog box with error information
        // and a button for aborting the job.
        StartOrResumeErrorProcedure(&fAbort, dwRet, ...);
    }
    // update write parameters
    pBuffer += dwNoOfBytesWritten;
    nNumberOfBytesToWrite -= dwNoOfBytesWritten;
};
```

PrtRead

Synopsis

```
DWORD WINAPI PrtRead (HANDLE hPrt, DWORD dwTimeout, DWORD *pdwType,
                      LPDWORD pdwParArray, LPDWORD pdwArraySize,
                      LPBYTE pbReadData, LPDWORD pdwReadDataLen)
```

Description

Printer messages are either automatically generated by the printer or are replies on application requests. The *PrtRead* function waits for messages from the printer and returns them.

If no message is available after *dwTimeout* milliseconds, the function will return without any information. Otherwise a message is returned in the parameters *pdwType*, *pdwParArray*, and *pbReadData*.

Each message can be identified by the returned *pdwType.* The other parameters are set dependent on the message. There are three types of messages.

- ESC [P1 ; … ; Pn <intermediate character><final character>[<STX><text><ETX>]

  This is the formal syntax of most printer messages.

P1 … Pn	are numeric values.
<STX><text><ETX>	is an optional data string.

  The numeric parameters *P1;…;Pn* after the control sequence introducer (CSI) *ESC [* are returned to the application in the numeric array *pdwParArray,* and the <*text*> between <*STX*> and <*ETX*> is returned in *pbReadData* for those messages which contain <*text*> at all.

- ESC I D <module ids/data ids/electric journal function enabled>

  This is the printer's reply to the command ESC I D. In this case the contents after ESC I D will be returned to the application in *pbReadData*. *pdwParArray* is empty.

Examples:

```
ESC I D $ M O D $ 0 4 0 1 2 5   0 1 0 5   B O O T P R O M .
P R M : $ M O D $ 0 3 1 2 1 9   0 2 0 8   4 9 1 5 _ S T D .
M O D ; $ M O D $   E L J

ESC I D $ M O D $ 0 4 0 1 2 5   0 1 0 5   B O O T P R O M .
P R M : $ M O D $ 0 3 1 2 1 9   0 2 0 8   4 9 1 5 _ S T D .
M O D ;
```

Note that the message always ends with ';' or ";MOD ELJ".

- ESC M O D, ESC m o d, ESC F N T, ESC BEL L

  For these messages, *pdwParArray* and *pbReadData* are empty. The message can be identified by the contents of *pdwType*.

There are two alternative ways of using *PrtRead*:

a) It can be used together with *PrtWrite* in one and the same transaction flow.

b) The other option is to use *PrtRead* in a separate read thread so that there are at least two threads that communicate with the printer: One thread for writing and one thread for reading and monitoring. In this case the required synchronization has to be done by appropriate synchronization objects like events or mutexes.

It depends on your application design which alternative is to be used.

Parameters

hPrt
Handle to the printer returned by *PrtOpen*.

dwTimeout
Maximum number of milliseconds to wait for information. The following defines have a special meaning:

PRT_IMMEDIATELY	return immediately
PRT_INFINITE	wait until information arrives

pdwType

The message identifier is returned here. The following identifiers are possible:

PRT_GLOBAL_STATUS	The global printer status is returned.
PRT_SPECIAL_MESSAGE	The special printer message is returned.
PRT_DOCUMENT_WIDTH	The width of the document is returned from the printer. This message is returned in response to the corresponding printer command.
PRT_MSR_MICR_READ_DATA	The result of an MSR or MICR read command is returned.
PRT_MSR_WRITE_STATUS	The printer's response to an MSR write command is returned.
PRT_CONTROL_POINT	Control point is reached. The control point number is returned in pdwParArray[0]. This message is returned in response to the control point command.
PRT_CONTROL_POINT_2	Control point 2 is reached. The control point number is returned in pdwParArray[0]. This message is returned in response to the control point II command.
PRT_PRINTING_UNIT_PARAMETERS	The printing unit parameters are returned. This message is returned in response to the corresponding printer command.

The following messages are sent by the bootstrap loader. See the HighPrint Manual for how to activate the load function:

PRT_READY_TO_LOAD	Printer is ready for download. This message is returned in response to the ESC M O D command.
PRT_READY_TO_LOAD_XE	Printer is ready for download. This message is returned in response to the ESC m o d command (only 4915xe/xa).
PRT_READY_TO_LOAD_FNT	Printer is ready for downloading a font. This message is returned in response to the ESC F N T command.
PRT_MOD_ID	The printer returned the identifiers of all loadware modules. This message is returned in response to the ESC I D command.
PRT_BEL_L	The printer is switched into the load mode.

pdwParArray
The numeric parameters *P1;…;Pn* after *ESC [* are returned in pdwParArray. The number of returned parameters depends on the message. For details see the HighPrint manual.

pdwArraySize
Here the application informs the DLL about the array size of *pdwParArray* (i.e. the number of array elements). On return the number of numeric parameters written into *pdwParArray* is returned. If *pdwArraySize* is 0 or less than the number of parameters to return, the function returns with *ERROR_MORE_DATA* and *pdwArraySize* holds the number of required array elements.

pbReadData
The contents of the information between <*start of text*> and <*end of text*> (if any) or the contents after ESC I D is returned.

pdwReadDataLen
Here the application informs the DLL about the size of the read buffer *pbReadData.* On return it holds the number of bytes returned in *pbReadData*. If *pdwReadDataLen* is set to 0 by the caller, the function returns with the required buffer size and *ERROR_MORE_DATA*.

Example

```
HANDLE  hThread;
DWORD   dwThreadId;

hThread = CreateThread(NULL, 0,
                       (LPTHREAD_START_ROUTINE) AsyncReadThread,
                        hPrt, 0, &dwThreadId);

DWORD AsyncReadThread(HANDLE hPrt)
{
    DWORD  dwRet;
    DWORD  dwParArray[10];
    DWORD  dwArraySize;
    BYTE  *pbReadData;
    DWORD  dwReadDataLen;
    DWORD  dwTimeout;
    DWORD  dwType;
    BOOL   fThreadRunning = TRUE;

    dwTimeout = 1000;
```

```
    while (fThreadRunning)
    {
        pbReadData = NULL;
        dwArraySize = 10;
        dwReadDataLen = 0;
        dwRet = PrtRead(hPrt, dwTimeout, &dwType, dwParArray,
            &dwArraySize, pbReadData, &dwReadDataLen);

        if (dwRet == ERROR_MORE_DATA)
        {
            // allocate the required memory
            if (dwReadDataLen > 0)
            {
                pbReadData = (BYTE *)
                              malloc((size_t) dwReadDataLen);
                if (pbReadData == NULL)
                {
                    dwReadDataLen = 0;
                }
            }
            // retry the read
            dwRet = PrtRead(hPrt, dwTimeout, &dwType, dwParArray,
                            &dwArraySize, pbReadData,
                            &dwReadDataLen);
        }
        if (dwRet == PRT_NO_ERROR)
        {
            // handle message
            switch (dwType)
            {
               ...
            }
        }
        else
        {
            // in case of unexpected errors we abort the thread
            if (   (dwRet != ERROR_NO_MORE_ITEMS)
                && (dwRet != WAIT_TIMEOUT) )
            {
                ErrorProcedure(hPrt, dwRet, ...);
                fThreadRunning = FALSE;
            }
        }

        if (pbReadData != NULL)
        {
            free(pbReadData);
            pbReadData = NULL;
        }
    }
} // end of AsyncReadThread
```

Return Codes

In general, the return values of the HPRDH functions are Win32 error codes. Therefore we refer you to the Win32 SDK for error code descriptions. Special remarks on some error codes can be found below.

Besides the Win32 error codes there are a few HPRDH-defined error codes which can be identified in general by BIT29 (APPLICATION_ERROR_MASK) set.

Define	Code	Description
PRT_NO_ERROR	0	The operation completed successfully. (See also *PrtWrite* for special remarks.)
PRT_OPEN_NOT_DONE	0x20000001	The printer has not been opened.
PRT_CLAIM_NOT_DONE	0x20000002	The printer has not been claimed.
PRT_COMMUNICATION_BROKEN	0x20000003	*PrtWrite, PrtRead, and PrtClaim*: It is not possible to communicate with the printer (e.g. USB cable disconnected). If the code is returned by *PrtRead* or *PrtWrite* then call *PrtRelease*. Try to establish a new session by calling *PrtClaim* again.
ERROR_INVALID_HANDLE	6	The handle is invalid. If *PrtClaim* returns this code it may indicate that the spooler has been stopped. In this case try to reopen the printer again.
ERROR_NOT_ENOUGH_MEMORY	8	Not enough memory is available to process this command.
ERROR_NOT_READY	21	The printer is not ready yet or not ready any more. Possible reasons: The print spooler closed the corresponding port because the port has been changed between *PrtOpen* and *PrtClaim*. The desired control points couldn't be reached, because the printer is not online any more.
ERROR_BAD_LENGTH	24	*PrtOpen*: The printer name contains more than 256 characters.
ERROR_OUT_OF_PAPER	28	*PrtWrite*: The printer is out of paper (USB printer only, ERROR_TIMEOUT otherwise)
ERROR_MORE_DATA	234	*PrtRead*: More data is available.

Define	Code	Description
WAIT_TIMEOUT	258	The operation timed out. (*PrtRead*, *PrtClaim* and *PrtWrite*)
ERROR_NO_MORE_ITEMS	259	*PrtRead* is called with dwTimeout = PRT_IMMEDIATELY and no information is available.
ERROR_TIMEOUT	1460	*PrtWrite*: The timeout period expired.
RPC_S_SERVER_UNAVAILABLE	1722	The print spooler may have stopped.
RPC_S_CALL_FAILED	1726	The print spooler may have stopped.
RPC_S_CALL_FAILED_DNE	1727	The print spooler may have stopped.
ERROR_INVALID_PRINTER_NAME	1801	*PrtOpen*: The printer name is invalid.

Using the Device Handler from Visual Basic

The following is a small code example written in Basic, which is to illustrate the use of the device handler from Basic.

```
Private Declare Function pPrtOpenW Lib "HPRDH" ( _
   ByRef Handle As Long, _
   ByVal ptrPrinterName As Long) As Long

Private Declare Function pPrtClose Lib "HPRDH" ( _
   ByRef Handle As Long) As Long

Private Declare Function pPrtClaim Lib "HPRDH" ( _
   ByVal Handle As Long, _
   ByVal Timeout As Long) As Long

Private Declare Function pPrtRelease Lib "HPRDH" ( _
   ByVal Handle As Long) As Long

Private Declare Function pPrtWrite Lib "HPRDH" ( _
   ByVal Handle As Long, _
   ByVal ptrWriteBuffer As Long, _
   ByVal NumberOfBytesToWrite As Long, _
   ByRef NumberOfBytesWritten As Long) As Long

Private Declare Function pPrtRead Lib "HPRDH" ( _
   ByVal Handle As Long, _
   ByVal Timeout As Long, _
   ByRef pdwType As Long, _
   ByVal ptrParArray As Long, _
   ByRef AarraySize As Long, _
   ByVal ptrReadData As Long, _
   ByRef ReadDataLen As Long) As Long

Dim pHandle         As Long


rem OPEN AND CLAIM

Private Sub OPEN_PRT_Click()
    Dim prnname As String
    Dim ptrname As Long
    prnname = "Wincor Nixdorf HighPrint 4915"
    ptrname = StrPtr(prnname)

    If pPrtOpenW(pHandle, ptrname) > 0 Then MsgBox ("OPEN FAIL")
    If pPrtClaim(pHandle, 5000) > 0 Then MsgBox ("CLAIM FAIL")
End Sub

rem CLOSE AND RELEASE

Private Sub CLOSE_PRT_Click()
    If pPrtRelease(pHandle) > 0 Then MsgBox ("RELEASE FAIL")
    If pPrtClose(pHandle) > 0 Then MsgBox ("CLOSE FAIL")
End Sub
```

```
rem WRITE DATA

Function WriteData(sbuffer As String)
    Dim buffer()        As Byte
    Dim StrBuffer       As Long
    Dim NumbOffWrite    As Long
    Dim StrLength       As Long
    buffer() = StrConv(sbuffer, vbFromUnicode)
    NumbOffWrite = 0
    StrBuffer = StrPtr(buffer())
    StrLength = Len(sbuffer)
    Status = pPrtWrite(pHandle, StrBuffer, StrLength, NumbOffWrite)
End Function


rem SAMPLE EJECT COMMAND

Private Sub Eject_Click()
    WriteData (Chr$(12))
End Sub


rem READ DATA

Function ReadData()
    Dim ptype       As Long
    Dim daten(30)   As Long
    Dim arraysize   As Long
    Dim ReadDataLen As Long
    Dim pararrysize As Long
    Dim readdatap   As Long
    Dim dataf       As String
    Dim pararray    As Long

    Erase daten()

    dataf = Space(512)
    arraysize = 30

    pararray = VarPtr(daten(0))
    readdatap = StrPtr(dataf)

    Do
           DoEvents
           ReadDataLen = 512
           arraysize = 30
           Status = pPrtRead(pHandle, Timeout, ptype, pararray, _
                             arraysize, readdatap, ReadDataLen)
    Loop
End Function
```

Sample Application

The SDK includes a code sample that demonstrates the usage of the API provided by HPRDH.DLL. Especially it shows how to use PrtRead() asynchronously in a second thread.

The transaction flow is as follows:

– The demo application asks for the printer name (e.g. "`Wincor Nixdorf HighPrint 4915")` to be able to open the printer. After open with *PrtOpen* and claim (*PrtClaim*), the command "Eject Paper" is sent to the printer and a control point is set using the *PrtWrite* function.

– The sample application waits until the control point is reached (using the Win32 functions *CreateEvent*, *WaitForSingleObject* and *SetEvent*) and the desired information is returned from the printer.

– Now the user is requested to enter paper. The application waits for paper to be inserted. For this it waits for the global printer status "PAPER".

– If the user presses the cancel button instead of entering paper, the application finishes calling *PrtRelease* and *PrtClose*.

– If paper is inserted, some lines and a form feed (FF) are written to the printer.

– After all writes are done, the application waits until the state "NO PAPER" occurs. Then the user is asked to load paper again to be able to print the next page. The application waits for the "PAPER" state before the next page is written.

– At the end of the application the printer is asked for the printing unit parameters. These parameters are shown in a window.

Please note that the sample application is written with a reasonable error handling.

Logging and Trace

Errors and warnings are written into the Windows application event log.

Additionally the driver software system provides trace functionality which can be activated by the following parameters in the registry. The parameters reside in the registry root of the language monitor. That is:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Print\Monitors\HighPrint

Parameter	Type	Description
TraceLevel	REG_DWORD	Trace level for device handler HPRDH
LmTraceLevel	REG_DWORD	Trace level for language monitor
		0 = No trace active 1 = Errors and warnings 2 = Errors, warnings and low information level info 3 = Errors, warnings and medium information level 4 = Errors, warnings and high information level 5 = Level 4 extended by a complete data trace in hex format

Please note that a debugger or debug viewer is necessary to see the trace output.

Abbreviations

API	Application Programming Interface
CD	Compact Disk
cpi	characters per inch
CSI	Command Sequence Introducer
DLL	Dynamic-Link Library
dpi	dots per inch
e.g.	exempli gratia (for example)
GDI	Graphical Device Interface
HW	hardware
i.e.	id est (that is)
LED	Light Emitting Diode
MICR	Magnetic Ink Character Recognition
MSR	Magnetic Stripe Reader
OCR	Optical Character Recognition
PnP	Plug and Play
pt	point (a unit used to measure the size of a character font)
SDK	Software Development Kit
USB	Universal Serial Bus

Literature

This manual is a part of the documentation series for the HighPrint printers.

The following HighPrint manuals are available:

- Programming Guide
- Paper Specification
- Operating Manual

Notes

Published by
WINCOR NIXDORF International GmbH
33094 Paderborn
Germany

Printed in Germany

Order No.: January 2008