soft Xpansion

Print to Document SDK Guide

Technology of virtual printers and universal document format, version 7

Contents

Introduction

As digital documents spread and drive out the traditional paper ones, the quantity of digital document format grows and also the range of their functions and advantages widens. Great number of advantages of digital documents is undeniable and overwhelming comparing to the paper documents.

But digital documents have one significant disadvantage, and no one knows who, when and how will be able to deliver them from this disadvantage. As there is no universal language for writing paper documents, so as there is no universal format for content storage of digital documents. The problem is that the digital document cannot be used even if you have a computer but do not have the necessary software. Some programs support 2, 5, or even 10 document formats but in reality there are countless digital document formats.

Of course, no one needs to use all existing document formats. But usually even a home computer have a few of programs installed for working with text documents, bitmap images, PDF documents, Word documents, Excel tables, PowerPoint presentations, etc. This list is endless. On an enterprise this list will be even longer, including standard document formats for some highly specialized professional areas and also irregular internal document formats used by enterprise in-house developed programs only within the company.

Sooner or later almost every enterprise faces a problem when there is some software's functionality does not allow performing some actions with document's content, such as converting the document into another format or converting a few documents with different formats into one document.

Universal Document Format

Despite a wide range of document formats and their purposes, almost every software supports a function of printing their documents. Some applications even cannot save some information in the document but can print this information. For instance, software that performs calculations and shows the results in form of tables, diagrams, etc. Or software that calculates 3D scenes and objects with set parameters in form of a general image. Or software that works with variable data base content and creates reports for a specific period of time.

Presence of functionality of printing any supported document formats or internal data makes it almost the only way for any software to perform data exporting in universal format, because the printing process is defined by the operating environment. Any information or data assigned for printing is transferred to the printing subsystem of the operating system in a single standardized format.

Unfortunately, such universal "format" has two serious disadvantages; although not for all tasks they are critical. However, as it was mentioned earlier, there is no alternative yet.

First disadvantage - by this way only visual information can be exported. Hidden elements, even if they are visual and were present in the original document, will not be transferred along with the exported data. Hyperlinks and other elements of the logical document structure will not be exported. Video (but the first frame) or audio information will not be exported too.

Second disadvantage - exported information is always split on pages, and their content is always has a fixed position. For example, voluminous web-page during printing will be divided on pages and their content will be fixed on those positions that were set according to the page width.

So, you face a task of getting data for one or few programs in universal format. You don’t want or cannot change those programs. The programs can print but you need digital documents and not paper ones. What’s to be done?

Print to Document

7th Generation

First steps towards "Print to Document" technology were made by soft Xpansion Company in 2002, and by the end of the year this technology was included in "PDF's leicht gemacht" software. This product used the technology for creating PDF files by printing on the virtual printer form any program.

Form that time the technology has been constantly and dynamically advancing. Almost every year a new version of **Print to Document** is being released. First versions of the technology were used by different products developed by **soft Xpansion** Company. Starting from the 5th version the technology is available for developers from other companies as a stand-alone product.

Since the first version was released there have been released a score of products that use this technology. Quantity of copies installed by final users is beyond a million. At the same time the quality of the technology is such that for the whole period of the technology existence there practically have been none calls from the customers to the support service with problems concerning **Print to Document**.

In the 7th version the technology has experienced an architecture modernization and the most significant changes from its creation. The reason for these changes is not in internal defects but in system requirements that have been significantly changed since its creation. OS Windows 95, Windows 98, Windows ME and Windows NT have fallen into history. Actual version of **Print to Document** does not support these versions of Windows and got rid of limitations that were caused by them. That makes the technology more effective and flexible.

Windows 7, Windows Vista and **Windows Server 2008** have come out and become more widespread. Windows versions that use **64-bit CPU architecture** actively increase the number of installed copies not only in business sphere but also in home and small business sector. Actual version of **Print to Document** is fully adapted for working with such versions of operating systems. It is also adapted for functioning in a network with computers that use different operating systems.

Without going deep into technical details (we will talk about it later) it is worthy to mention that functional abilities of the current version of the technology are wider and work more effective comparing to the previous versions. At the same time, gathered Know-how and experience of developing the technology itself and products on its base allows enlarge its functionalities without loosingreliability - the main quality of the technology.

Virtual Printer

Print to Document technology allows installing virtual printers on computers with OS Windows.

These printers are called virtual because of the two reasons:

- such printer does not require any physical device in order to function
- such printer does not really print anything

Its main function is to "deceive" a software that can print and receive the necessary data form it in universal digital format.

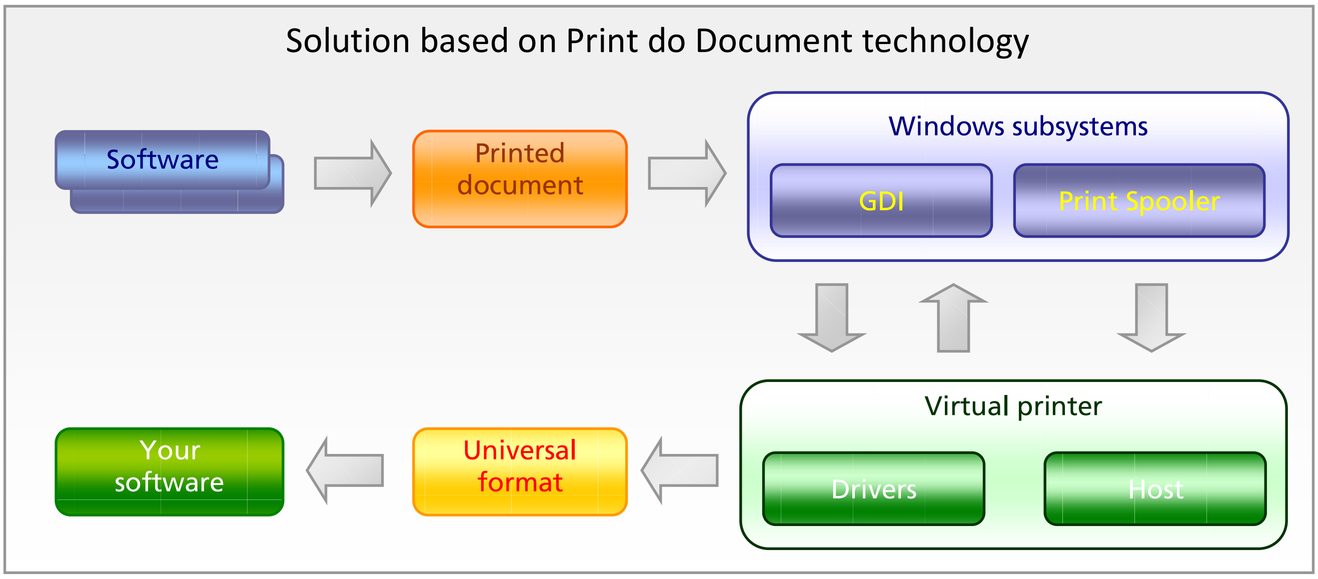
Despite its virtuality such printers are not different from real devices. After a virtual printer installation even the operating system will not see the difference between a virtual printer and a real device. The printer will be placed into the standard "Printers and Faxes" folder. That means that an end-user will not have difficulties using a virtual printer (f the **Print to Document** software will include the function of using the virtual printer by end-users).

Since the operating system will not see the difference between a real and a virtual printer, so as any software that will print on a virtual printer will not see the difference too. Any additional adaptation or setting up is not needed; only choose the necessary printer from the list of available printers on the computer.

After the necessary in formation is sent to a virtual printer and has been received by it in a universal format, its main function is completed. After that the virtual printer passes received information to the necessary software.

Solution Architecture

The bellow scheme shows how **Print to Document** software can receive data from practically any other software in universal format.



Printing in Windows never performs strait but through special subsystems of the operating system: graphic subsystem (GDI) and Print Spooler subsystem.

GDI subsystem provides applications with a standard printing interface (Print API) that allows standardizing data output during printing from any software. It also helps avoiding the necessity of taking into account specificities of printing devices.

During printing process GDI communicates drivers of the chosen printer by special command system in order to create a "print job" - printed data in universal format. This data as a Spool File is being transferred to the Print Spooler subsystem which passes it to the special driver from the printer's set to the Print Processor. If printing is performed on a network printer, Print Spooler transfers the Spool File via the network to that computer where the network printer is installed. By this, the data not only is being converted into universal format but also is being transferred from the network to a single place.

Components of the Technology

Print to Document technology set includes 3 **drivers**: configuration driver, rendering driver (both are implemented in one module) and print processor. The set also contains a library for driver and printer installation. The last component of the technology may be used optionally if needed. This component is a special host-program that allows print job processing to be done in its own context and not in the system Spooler's context.

The **configuration driver** defines a program interface that communicates the printer's configuration component in your application, which provides a user interface component that enables users to control a printer's selectable options. It also enables reacting on main events that acquire in the driver during printing on the virtual printer (for example, "CreateDC", "StartDoc", etc.).

The **rendering driver** converts the graphics commands (printed data) from the application to universal data format as an enhanced metafile (EMF) file that can be sent to the Print Spooler as print job.

The **print processor driver** has already been called by Print Spooler and receives print job data and additional job information. This driver defines a program interface that communicates the processing component in your application (connector), which provides the necessary logic for received data processing in universal format (EMF files).

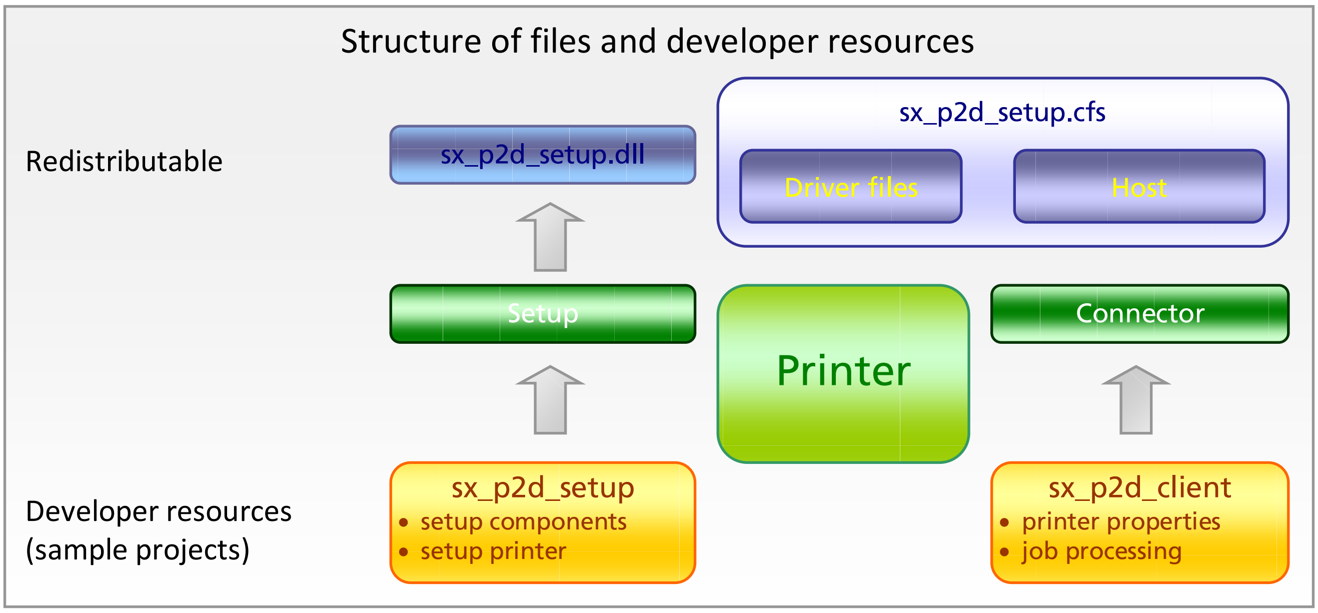
The **install library** provides methods for installation and uninstallation of all components of the **Print to Document** technology in the operating system. It is also used for installation, uninstallation and configuration of one or few virtual printers.

The **host program** may be used optionally for three purposes:

The first - Print Processor is always activated by Print Spooler which is launched in the system account context. Thus the activation of your application's component (connector) for print job processing will also be launched in the system account context. But this way sometimes is not usable. For example, if interactive communications need to be performed with end-user, or if a print job has to be processed in such user context that was used for application to print the data on the printer. In this case, Print Processor will launch a special process (host) in the context of the necessary user, and then this process will activate the connector for print job processing. In some occasions it is effectively to use host program for print job processing under the system account in order to avoid inerblocking of Print Spooler which sequentially transfers print jobs to the Print Processor.

The second task is multithreading, parallel print job processing. Print Spooler works with queue of print jobs only sequentially and sometimes the processing of some jobs can take quite a long time. Because of that during mass document printing and jobs processing on computers with few processors, the host allows the printing process to be much more effective.

The third task is saving print jobs received by Print Processor in temporary files instead of notifying and activating the connector for their processing. In this case, in the time you need your application can ping the host-process through a special interface for unprocessed jobs and process them by itself.



And finally, the installation set contains two sample projects. Both samples are written in C++, but 'Print to Document' technology can be used from any language that supports COM interface.

sx_p2d_setup project shows how to use technology's components and printers by using setup components.

sx_p2d_client project shows how to implement the connector between the printer and your software - organization of print jobs processing and creating of its own printer's property dialog and other printing options.

Folders and Files in SDK

DevRes - this folder contains a header and IDL files that a developer needs in order to integrate Print to Document into his own software. Also the folder contains the project samples.
Docs - technology documentation.
Redist - files that need to be added to the software installation set for redistribution to end-user computers.
Trial - trial printer installation set. After its installation you can evaluate its work straight away.

System Requirements

Windows platform: **Windows 7, Windows Vista**, Windows XP,
Windows 2008, Windows 2003, Windows 2000.

Processor architecture: 32-bit, **64-bit**.

VMWare and Remote Desktop Session is supported.

Licensing and Pricing

The object of the license is an instance of the virtual printer which can be installed on any number of computers, delivered in a set of any number of products and under any name. On one computer it can be installed in one copy irrespective of the printer's name.

License or condition	Price (€) or bidding (%)	Notes
Professional license, 1 instance	2900,-	one-time price
Business license, 1 instance	4900,-	one-time price
Second instance	50%	
Third and more instances	25%	per instance
Network printer	25%	
World wide	25%	
Support	25%	year price

Professional license permits the usage of the printer for printing from any applications in a non-massive way (< 30 print jobs per hour), i.e. the ultimate user usually prints directly on the printer.

Business license the same as Professional license, but there is no limit for the print jobs number.

Printer and Component Setup

SDK contains **sx_p2d_setup.dll** which provides a developer a set of methods for installation, configuration and uninstallation of other SDK components and virtual printers. Because of that, the whole installation process requires activation of only few methods with the necessary configuration parameters. The component includes IP2DSetup70 COM interface and duplicates its methods by exporting C-functions. By this way the developer can use this component from applications written in different languages (including .NET languages) and also in scripts of different setup programs (MSI, InstallShield, etc.).

SDK also includes a sample written in C++ which uses the setup component. The sample demonstrates a typical installation, configuration and uninstallation of the **Print to Document** virtual printer with a trial license. After this trial printer will be installed, you can evaluate its work by printing on it from any program. Of course, the processing of printed pages is practically disabled and includes only the ability of viewing of printed pages in a dialog. After closing the dialog the printed pages will not be saved.

The setup component does not require any software and uses only Windows system functionalities. For proper work of the setup component it is required to have administrative rights for user during working with this component. In other words, the user has to be included in the group with administrative rights. For successful work in Windows Vista and the later versions with UAC activated there is an additional requirement - an application that uses the setup component should have a manifest that defines the necessity of administrative rights for this application. An example of the setup application that is included into SDK has the described manifest.

Before the setup component usage, copy its file sx_p2d_setup.cfs on the hard drive and register it in a standard way as a COM component.

IP2DSetup70 Interface

This interface provides the developer with a set of methods for installation, configuration and uninstallation of SDK components and virtual printers.

OpenLibrary([in] BSTR sResPath);

The Setup component initialization before its usage.

sResPath - actual path to the **sx_p2d_setup.cfs** file delivered along with the setup component.

CloseLibrary();

The setup component uninstallation.

BSTR GetVersion();

Print to Document driver's version in CFS file.

The version will be returned as string, for example, "7.1.3.0".

BSTR GetDriverVersion([in] BSTR Name, [in] boolean b64);

Print to Document driver's version which is installed on the computer (if installed).

The version will be returned as string, for example, "7.1.0.0" or "0" if driver in not installed.

Name - Print to Document driver's name.

b64 - if TRUE, then return 64-bit version of the driver, otherwise - 32-bit version.

BSTR GetDriverProcVersion([in] BSTR Name);

Print processor's version which is installed on the computer (if installed).

The version will be returned as string, for example, "7.1.0.0" or "0" if driver in not installed.

Name - Print processor's name.

InstallDriver([in] BSTR Filename, [in] BSTR Name, [in] boolean b32, [in] boolean b64);

Print to Document driver's installation. If the driver is already installed but has an older version, then the driver will be updated. After updating it is required to restart Windows.

Filename - drivers' filename. With such filename the driver will be installed.

Name - drivers' name. With such name the driver will be installed.

b32 - if TRUE, then 32-bit driver will be installed.

b64 - if TRUE, then 64-bit driver will be installed.

UninstallDriver([in] BSTR Name);

Uninstalls Print to Document driver. After uninstall it is required to restart Windows.

Name - driver's name.

InstallDriverProc([in] BSTR Filename, [in] BSTR Name);

Print processor's installation. If the driver is already installed but has an older version, then the driver will be updated. After updating it is required to restart Windows.

Filename - Print processor's filename. With such filename it will be installed.

Name - Print processor's name. With such it will be installed.

UninstallDriverProc([in] BSTR Name);

Uninstalls Print processor. After uninstall it is required to restart Windows.

Name - Print processor's name.

InstallPrinter([in] BSTR Name, [in] BSTR PrinterID, [in] BSTR DriverName, [in] BSTR PrintProcessorName, [in] BSTR PortName, [in] BSTR ShareName, [in] BSTR LicensePath);

Virtual printer's installation.

Name - printer's name.

PrinterID - printer's ID. It should correspond to the ID in the license file.

DriverName - Print to Document driver's name, which is installed by the "InstallDriver" method.

PrintProcessorName - Pint processor driver's name, which is installed by the "InstallDriverProc" methid.

PortName - port name, which is used by the virtual printer, usually "LPT1".

ShareName - printer's share name. If the name is entered, the printer will be shared and available in a network. If NULL, the printer will not be shared. In order to use the printer as a network printer, the special permissions are required in the license file.

LicensePath - actual path to the license file.

UninstallPrinter([in] BSTR Name);

Deletes the virtual printer.

Name - name of the printer.

InstallPrinterDependence([in] BSTR PrinterName, [in] BSTR SrcPath, [in] BSTR DstFileName);

Registers a file required for using the printer in the network printer mode. During adding the network printer, Windows will transfer the registered files from the computer with shared printer to the computer where the network printer is being added; generally, it is a module that implements IP2DEvents70 interface. If necessary, it is allowed to register few files in this way; for example, files with settings or resources for the interface realization. All such files are being copied in a special folder.

PrinterName - printer's name.

SrcPath - actual path to the file that is being registered,

DstFileName - the file name with which it will be copied to the system folder and transferred to the remote computers.

UninstallPrinterDependences([in] BSTR PrinterName);

Deletes all registered files.

PrinterName - printer's name.

BSTR GetPrinterRegKey([in] BSTR PrinterName);

Retrieves the name of the registry key under HKEY_LOCAL_MACHINE, which contains printer settings and properties.

PrinterName - printer's name.

BSTR GetPrinterRegKeyByID([in] BSTR PrinterID);

Retrieves the name of the registry key under HKEY_LOCAL_MACHINE, which contains printer settings and properties.

PrinterID - printer's ID.

BSTR GetPrinterRegUserKey([in] BSTR PrinterID);

Retrieves the name of the registry key under HKEY_CURRENT_USER, which contains printer settings and properties.

PrinterID - printer's ID.

SetupPrinterEventHandler([in] BSTR PrinterName, [in] BSTR CoClassGUID32, [in] BSTR FileName32, [in] BSTR CoClassGUID64, [in] BSTR FileName64, [in] unsigned long EventMask);

Configures a virtual printer for usage of the components that implements IP2DEvents70 interface. The main task for this component is a product orientated printer properties.

PrinterName - printer's name.

CoClassGUID32 - GUID of the class that implements P2DEvents70 interface. Module that contains this class has to be compiled on the 32-bit platform.

FileName32 - file name of the class compiled for the 32-bit platform. Can be NULL for those printers that will not be used in a network mode. For network printers such file should be registered by "InstallPrinterDependence" method.

CoClassGUID64 - GUID of a similar class for 64-bit platform. Can match the GUID for 32-bit class.

FileName64 - file name of the class compiled for the 64-bit platform. Can be NULL for those printers that will not be used in a network mode. For network printers such file should be registered by "InstallPrinterDependence" method.

EventMask - event mask. For details, refer to the description of IP2DEvents70 interface.

```
InstallConnector([in] BSTR PrinterName, [in] unsigned long ConnType, [in] BSTR CoClassGUID, [in]
BSTR HostPath, [in] boolean Host64);
```

Registers a connector to the virtual printer component for processing of print jobs. If needed, installs the host application.

PrinterName - printer's name.

ConnType - type of connection between the print processor and the connector component. For details, refer to the description of the IP2DConnector70 interface.

CoClassGUID - GUID of the class that implements IP2DHSJobProcessor70 interface if ConnType = **P2D_CONN_DIRECT**, otherwise the class should implement IP2DConnector70 interface.

HostPath - actual path to the place where the hosed application will be placed. Should be NULL if ConnType = **P2D_CONN_DIRECT**.

Host64 - if TRUE, then 64-bit version of the host application will be installed, otherwise - 32-bit version that works on 32-bit and on 64-bit platforms.

```
UninstallConnector([in] BSTR PrinterName);
```

Uninstalls the connector component.

PrinterName - printer's name.

Getting Started

A few rules and recommendations on how to work with the **Print to Document SDK** and the setup component.

1. Included in SDK trial license allows installing of virtual printers and testing integration with client software only on developer's computers and only for evaluating purposes. Using the trial license for any other purposes, including virtual printer demonstration within client software is prohibited.

2. **Print to Document** technology is used by different products and software developers. In order to avoid technical conflicts between such products, using names, identifiers and constants that are declared in the "trial.h" file and are used in the installation procedure of the trial printer within SDK is prohibited. During integration with own software the developers have to replace values of such elements with the different ones in order to guarantee their uniqueness.

Also you should replace with your own unique values class's GUIDs that are implemented by IP2DEvents70 and IP2DConnector70 interfaces:
{7194F58E-699F-4C84-9018-8BDB245C2A30}
{8D6C6CDF-5E47-42DE-9F6B-5B288AD87534}

in **sx_p2d_client** sample.

3. Trial license does not allow printer and connector integration in **P2D_CONN_DIRECT** and **P2D_CONN_PASSIVE** modes.

4. During printer installation on a computer with Windows 2000 or Windows XP (without SP or SP1) you can install only 32-bit drivers. 64-bit drivers can be installed starting from Windows XP SP2. Because of that, using network virtual printer on computers with 64-bit OS Windows can be possible only if shared printer is installed on a computer with Windows XP SP2 or later version.

Printer Events and Configuration

Straight after installation of Print to Document components and virtual printer, the printer is ready to use. All its settings are optional and are not required for its full work. This section of the documentation contains detailed technical information concerning all nuances of printer configuration and its drivers and can be difficult to read for not experienced SDK users.

We will start from explaining that from different point of view the meaning of "printer" is different.

From end-user's point of view a "printer" is a device that stands on the desk or in a near room and has a bad habit of finishing a pack of paper always earlier then we would expect and in inconvenient time ☺. Each such device has an icon associated with it in the "Printers and Faxes" folder and in the "Print" dialog practically in every program. Since **Print to Document** technology does not use a physical device, so the end-user is delivered from all problems associated with it and has to deal only with printer icons in software.

From the Windows' point of view a "printer" is a system object registered in a specific way in the system. This object has a few drivers integrated in the Graphics Device Interface that allow printer to function. Windows strictly regulates printer's registration process, its functioning and configuration, and also the format and methods of integration of the Graphics Device Interface with the printer's drivers.

From the applications and their developers' point of view a "printer" is specific object in the Graphics Device Interface (GDI). This subsystem generates its fundamental objects for the printer - Device Context (DC) or Graphics in GDI+ ideology. The application and its developer have the ability of using the printer object and DC objects. They can manipulate these objects only by programmed graphics interface which specified and documented by Windows' developers. Technically, Windows does not forbid using printer's drivers directly but Print to Document technology does. First of all, because that would contradict the main idea of the technology - forming the data in universal format; this task is resolved by Windows GDI. Second, such way of printers' usage would be successful only between a printer and an application "acquainted" with each other, i.e. from the same developer. Because of specific properties of printer objects, the Windows GDI provides the applications with specialized abilities of their configuration.

The difference between Windows' and applications' points of view concerning the definition of a "printer" leads to serious technical consequences. An application can receive a printer object only by requesting it from GDI. This subsystem, after the first application's call to any available printer on the computer, creates such object in application's process context and configures it according to the settings received from the system printer object registered in Windows (in the system part of the registry).

The results of such approaching are:

- some printer properties can be configured only in the system object. It should be done only in the registry. The changes in printer properties will be available only in those applications that will be launched after the changes were made. In other words, only in those applications that will call the printer for the first time after its system object was changed;
- printer configuration by the programmed graphics interface will change only the properties of the printer's dynamic object in context of the current application and will not affect the system object configuration or dynamic objects of this printer in other processes.

Print to Document technology significantly widens the set of configurable printer's characteristics and properties that are specified by Windows. It also provides unique abilities in manipulating with Windows specified properties for traditional printers. For example, a paper sheet size for real printers is always limited with A4 or A3 format for office printers. Virtual printers will allow using practically any size. Real printer have fixed printed margins (usually, they are equal zero). Virtual printers allow configuring the margins in the way you need.

Another significant advantage of **Print to Document** technology is the presence of an engine that allows processing of key evens during the printing process, implementation of specific product-orientated properties that are available for end-user and providing helpful information associated with every print job. In order to implement such engine, the IP2DEvents70 COM interface has to be implemented into the software that contains a virtual printer. If there is no need for solving such tasks then implementation of the interface is not required.

Printer Configuration in Registry

Configuration of printer's system object is stored in the registry. The name of the key is specified by the Microsoft developers and depends on Windows version, name of the printer, printer type (local or network printer). Because of that, the developers are advised to use the name of the key which is provided by P2D SDK methods.

Printer registry key is placed in HKEY_LOCAL_MACHINE section of the registry. And because of that, the access to the key from applications launched from user's context with restricted rights will be read-only. So, it is impossible to change the configuration of printer's system object from such applications.

Print to Document technology solves this problem. During creation of dynamical copy of printer object in application (after calling Windows GDI), printer drivers are trying to read printer's characteristics and properties from the registry key placed in HKEY_CURRENT_USER. If it is successful, the drivers replace corresponding characteristics and properties in the system object. Access to the keys in this section of the registry is usually unrestricted, and the applications are allowed to configure the printer even in the restricted user's context.

To allow work in such mode, a special property should be set (refer to the "UserKey"), otherwise used-users with restricted rights will not be able to change printer configuration.

The described engine also allows setting a different printer configuration for different users on the same computer. This will not affect parameters placed in subkeys of the main registry key ("Papers", "Resolutions", "Events" and "Connector").
The full name of the key can be received using methods of the setup component: GetPrinterRegKey, GetPrinterRegKeyByID and GetPrinterRegUserKey. And also by using the methods of an additional IP2DDialog70 interface in the printer property dialog.

Registry Key Structure and Values

PrinterID (value)

The printer identifier is set during the installation and cannot be changed during the work process. If the value of this parameter will be different from the corresponding value in the license file, the printer will be blocked.

UserKey (value)

The parameter enables or disables printer configuration by using registry key placed in HKEY_CURRENT_USER section by users, including those with restricted rights.
Parameter type is a number. Default value is "0"; none-zero value enables configuration in HKEY_CURRENT_USER.

Shareable (value)

The parameter enables or disables printer usage as a network printer. Windows cannot forbid network printer installation if the local printer is shared. So, this parameter does not block the installation of a network printer but disables printing on such printer.
Parameter type is a number. Default value is "0"; none-zero value enables a printer to function as a network printer.

If enter a printer's share name during the installation, the parameter will be set as "1". In order to use a printer as a network printer, the special permissions in the license file are required.

ResetDC (value)

This parameter is used for resolving a problem caused by the System Spooler's work specificity (system printing service in Windows) with some programs such as Notepad (included in Windows). During printing from this program, spooler does not call the DevMode structure, which contains the current printer's configuration, from the printer driver and associates the older version of the structure with the print job.
Parameter type is a number. Default value is "0"; none-zero value during DC printer creation enables the driver automatically call the system function ResetDC() that directs the spooler to require the actual state of DevMode.

Fields (value)

The parameter enables additional supported fields of the printer driver (see **DEVMODE** structure, **dmFields** member).
Parameter type is a number. Default value is "0". The following values are allowed:
DM_COLOR, DM_SCALE, DM_COPIES, DM_COLLATE, DM_DUPLEX. Values can be combined.

Scale (value)

The parameter specifies the factor by which the printed output is to be scaled. The apparent page size is scaled from the physical page size by a factor of Scale / 100. For example, a letter-sized page with a Scale value of 50 would contain as much data as a page of 17- by 22-inches because the output text and graphics would be half their original height and width. DM_SCALE value must be enabled in the **Fields** registry value.
Parameter type is a number. Default value is “100”. Value must be in a range 1..10000.

MaxCopies (value)

The parameter defines the maximum number of copies if the printer supports multiple-page copies. DM_COPIES value must be enabled in the **Fields** registry value.
Parameter type is a number. Default value is “1”.

Color (value)

The parameter defines type of the printer - color or black&white. If black&white value is selected then during printing process will not be any automatic transformations performed, and the printed data usually remains in color. But some programs could be orientated on printer’s type and print the data differently. DM_COLOR value must be enabled in the **Fields** registry value.
Parameter type is a number. Default value is “0”; none-zero value means color printer.

HorizontalResolution (value)

The parameter defines printer’s horizontal resolution, DPI.
Parameter type is a number. Default value is “300”.

VerticalResolution (value)

The parameter defines printer’s vertical resolution, DPI.
Parameter type is a number. Default value is “300”.

PaperID (value)

The parameter defines a paper type used by the printer. The type defines the paper’s size. Values from ‘0’ to ‘255’ are reserved by Microsoft for standard papers. Additional sheet sizes and paper identifiers can be entered in the “Papers” subkey. Paper ID = 256 is displayed in print dialogs in the list of papers as “Custom size”. A printing program should set a necessary sheet size when choosing this paper type.
Parameter type is a number. Default value is “256”.

PaperWidth (value)

Sheet’s width, if ID = 256 (custom size) is selected. The value is measured in 0.1 mm.
Parameter type is a number. Default value is “2100”.

PaperHeight (value)

Sheet’s height, if ID = 256 (custom size) is selected. The value is measured in 0.1 mm.
Parameter type is a number. Default value is “2970”.

Landscape (value)

Sheet’s orientation.
Parameter type is a number. Default value is “0”; none-zero value means a landscape orientation.

PaperSource (value)

This parameter is for localization of corresponding printer’s property.
Parameter type is a string. Default value is “Auto source”.

LeftMargin (value)

Printer’s left margin (a none-printed zone). The value is measured in 0.1 mm.
Parameter type is a number. Default value is “0”.

RightMargin (value)

Printer’s right margin (a none-printed zone). The value is measured in 0.1 mm.
Parameter type is a number. Default value is “0”.

TopMargin (value)

Printer’s top margin (a none-printed zone). The value is measured in 0.1 mm.
Parameter type is a number. Default value is “0”.

BottomMargin (value)

Printer’s bottom margin (a none-printed zone). The value is measured in 0.1 mm.
Parameter type is a number. Default value is “0”.

ClientData (value)

The parameter contains a product orientated data and settings. For example, for printer that creates PDF files it could be the necessary PDF file properties that are entered by an end-user in the printer configuration dialog. During printing process, the driver adds actual values of this parameter to each print job that the software which is processing a print job could use it. For details, refer to the Job info (Client data).
Parameter type is binary. Maximum size is 4096 bytes.

Papers (subkey)

This key can have a few subkeys each of which defines product orientated paper size that an end-user can use for printing on the virtual printer. The name of each subkey is used as a paper name in print dialogs. For example, “Papers\A0” printer subkey will add to the printer’s list of papers a paper with “A0” name.

Each of these subkeys has to have tree parameters: PaperID, PaperWidth and PaperHeight. PaperId has to be unique in the list and be non-zero. Values from 1 to 256 can be used only for localization of standard paper types and a special (256) “Custom size” paper type. If PaperID is more then 255, PaperWidth and PaperHeight parameters are required, otherwise they are ignored. The sizes are measured in 0.1 mm.

Resolutions (subkey)

This key can have a few values, each of which will define supported printer resolution (DPI). Printing program can choose from the list of supported resolutions the most applicable for actual task. Some programs can let an end-user to select the resolution. The name of each subkey is not important because only its value is used.

Events (subkey)

This subkey configures the printer’s driver and contains parameters for its integration with the software that has a virtual printer. Such integration is not required for virtual printer, so the key will be filled only when the “SetupPrinterEventHandler” method of the setup component will be called during the printer installation.

CoClassGUID32 (value)

GUID of the class that implements IP2DEvents70 interface. Module that contains this class should be compiled on 32-bit platform.

FileName32 (value)

Name of the file which contains a class compiled for 32 - bit platform.

CoClassGUID64 (value)

GUID of a similar class for 64 - bit platform. Can match the GUID of the class for 32 - bit platform

FileName64 (value)

Name of the file which contains a class compiled for 64 - bit platform.

EventMask (value)

Event mask, for details, refer to the IP2DEvents70 interface.

Connector (subkey)

This key is used for configuration of printer’s print processor. The key contains necessary parameters for its integration with software that has a virtual printer. This integration is essential, because the data in universal format is passed from the virtual printer to the following processing by this integration.
The key is being filled during printer installation after calling of "InstallPrinerConnector" method of the setup component.

ConnType (value)

Type of connection between the print processor and the connector component. For details, refer to IP2DConnector70 interface description.

CoClassGUID (value)

GUID of the class that implements IP2DHSJobProcessor70 interface if ConnType = **P2D_CONN_DIRECT**, otherwise the class should be implemented by IP2DConnector70 interface.

HostPath (value)

Actual path to the place where the hosed application will be placed.

Threads (value)

Quantity of threads in the host application. If the parameter is empty or zero then the effective quantity for the current computer is calculated. It is not used if ConnType = **P2D_CONN_DIRECT**.

JobPath (value)

Actual path to the folder where temporal print jobs will be stored if ConnType = **P2D_CONN_PASSIVE**.

Printer Driver Events and IP2DEvents70 Interface

This COM interface is used by the printer's configuration driver for connection with the software that includes a virtual printer. If necessary the software can implement this interface. If the interface has been implemented, the module which implemented it has to be registered in a standard for COM components way. Besides that, the interface is registered by "SetupPrinterEventHadler" method for the virtual printer.

For those printers that will function in a network mode, the virtual printer's module has to be registered by "InstallPrinterDependence" method. It is important to remember that if the module for its functioning requires other modules, components or resource files than each of these files has to be also registered by the same method. Obviously, this or other modules registered in this way do not have to require an additional installation or specific system components in order to function. Otherwise, they will not function on other computers where the virtual printer will be added as a network printer. On the computer where the network printer is being added, the configuration driver will provide COM registration of the module which implements IP2DEvents70 interface.

In order to provide a correct functioning of the virtual printer on the computers with 32- and 64-bit CPU architectures, two modules have to be compiled - one module per architecture. CoClass that implements the interface in modules for 32- and 64-bit platforms can have one and the same or different GUID.

During implementation of the interface, please, take into account that object which implements this interface is used by the printer's configuration driver which is loaded by any application that uses the virtual printer. It can be PrintSpooler that works in the system user context, other system service or any application that functions in any user account context (not only active) with the restricted rights.
Also take into account that the life cycle of the object that implements IP2DEvents interface is provided by the driver only for one event processing. Usually after event processing the object will be destroyed. It is not recommended to store any data in this object.
During interface registration in virtual printer, a limited set of events on which the driver will inform the object can be set; processing of unnecessary events can be disabled.

Events in the P2D Driver

P2D_EVENT_PRN_PROP

Virtual printer's property sheet opens. One or few property pages with a set of specific properties of the virtual printer can be added do the dialog.

P2D_EVENT_DOC_PROP

Virtual printer's property sheet opens. One or few property pages with a set of specific properties of the printed document can be added do the dialog (for example, document's PDF properties for PDF-printer).

P2D_EVENT_PRN_NET_ADD

Network printer has been added (installed). Usually, such computer does not have software with a virtual printer, so if some additional setup needed it can be done at this moment.

P2D_EVENT_PRN_NET_DEL

Network printer has been deleted (uninstalled). It is a good time for deleting temporally data and resources.

P2D_EVENT_DOC_CREATEDCPRE

The application has requested a printer's device context from the graphic subsystem; DC object has not yet created.

P2D_EVENT_DOC_CREATEDCPOST

The application has requested a printer's device context from the graphic subsystem; DC object has been created.

P2D_EVENT_DOC_STARTDOC

The application has called the "StartDoc" function. PrintSpooler is creating a print job.

P2D_EVENT_DOC_ENDDOC

The application has called the "EndDoc" function. PrintSpooler has created a print job and sends it to the print processor.

P2D_EVENT_DOC_DELETEDC

The application deletes printer's DC object.

I2PEvents70 Interface

```
HRESULT PrinterProperties([in] BSTR sPrinterName, [in] IUnknown* pDialog);
```

The method is called by the driver for P2D_EVENT_PRN_PROP event notification.

sPrinterName - printer's name.

pDialog - object that implements IP2DDialog70 interface; it is used for adding custom property pages into the printer's property dialog.

```
HRESULT DocumentProperties([in] BSTR sPrinterName, [in] IUnknown* pDialog);
```

The method is called by the driver for P2D_EVENT_DOC_PROP event notification.

sPrinterName - printer's name.

pDialog - object that implements IP2DDialog70 interface; it is used for adding custom property pages into the printer's property dialog.

```
HRESULT DocumentEvent([in] BSTR sPrinterName, [in] UINT_PTR hPrinter, [in] LONG Event);
```

The method is called by the driver for notification about the following events: P2D_EVENT_DOC_CREATEDCPRE, P2D_EVENT_DOC_CREATEDCPOST, P2D_EVENT_DOC_STARTDOC, P2D_EVENT_DOC_ENDDOC, P2D_EVENT_DOC_DELETEDC.

sPrinterName - printer's name.

hPrinter - printer's system handle.

Event - event type.

```
HRESULT PrinterEvent([in] BSTR sPrinterName, [in] LONG Event);
```

The method is called by the driver for notification about the following events: P2D_EVENT_PRN_NET_ADD, P2D_EVENT_PRN_NET_DEL.

sPrinterName - printer's name.

Event - event type.

I2PDialog70 Interface

```
BSTR GetPrinterKey();
```

Retrieves the name of the registry key under HKEY_LOCAL_MACHINE, which contains printer settings and properties.

```
BSTR GetPrinterUserKey();
```

Retrieves the name of the registry key under HKEY_CURRENT_USER, which contains printer settings and properties.

```
AddPropertyPage([in] BSTR sTitle, [in] ULONG_PTR hInstance, [in] BSTR sDlgTemplate, [in]
ULONG_PTR pDlgProc,[in] ULONG_PTR hPageKey);
```

Adds to the printer's property dialog a new property page.

sTitle - property page title.

hInstance - A handle to the instance from which to load the dialog box template, icon, or title string resource.

sDlgTemplate - Dialog box template to use to create the page. This member can specify either the resource identifier of the template or the address of a string that specifies the name of the template.

pDlgProc - A pointer to the dialog box procedure for the page. The dialog box procedure must not call the EndDialog function.

hPageKey - an identifier of the added property page; inputs into the dialog box procedure as "IParam" from "WM_INITDIALOG" notification.

Printer Configuration in Application (Specific for P2D Printers)

Printer configuration by using of Windows program interface changes only dynamic object's properties of the printer in current application context. It does not affect the configuration of system or dynamic objects of this printer in other processes. "DocumentProperties" function is used for printer configuration.

Print to Document virtual printers provide printing application with extended capabilities for managing the printer during the printing process. Besides the "DocumentProperties" function another standard Windows GDI function can be used - "ExtEscape". This function can be called by special codes that are supported by **Print to Document** printers.

Escape Codes

P2D_ESCAPE_RESET_DRV_DATA

"ExtEscape" call with such code will instruct the driver to update the printer's dynamic object from the registry. In order changes to be made also in DC objects, created before the update was performed, "ResetDC" function should be called for such objects; or these objects should be deleted and created again.

P2D_ESCAPE_RESET_PAPERS

Instructs the driver to update the list of supported papers from the registry.

P2D_ESCAPE_RESET_RESOLUTIONS

Instructs the driver to update the list of supported resolutions from the registry.

P2D_ESCAPE_SET_JOB_ID

Instructs the driver to associate a string identifier with the print job, for details, refer to the Job info section. The string identifier is passed by the "IpszlnData" parameters of "ExtEscape" function; it should be in Unicode format and less than 40 characters. Before "StartDoc" calling, "ResetDC" should be called in order for Print Spooler to update printer's actual configuration including the associated identifier. After "EndDoc" calling it is recommended to "clear up" the identifier by setting up an empty string as its value and calling "ResetDC", otherwise the Print Spooler can use cashed value for other print jobs.

P2D_ESCAPE_SET_CLIENT DATA

Instructs the driver to update a set of client's binary data (actual product-orientated printer settings) in printer's dynamical object in order to associate it with its print jobs. For details, refer to the Job info section. Associative process is analogical to the one that is described for P2D_ESCAPE_SET_JOB_ID.

Print job processing

Print Processor and Connector

Data in universal format (Spool EMF) that has been printed on the printer is transferred by a special system service (PrintSpooler) to the printer's print processor (specialized driver). The spool data of the print job is transferred by one call as one block (pseudo file). Besides this data, print processor receives another block of information (job info) which is associated with the print job and contains the additional information about it. A part of the data in this block is formed by the PrintSpoller service; contents and format or the data corresponds with JOB_INFO_2 structure defined by Windows Platform SDK. The second part of the data, during printing on the virtual printer from any software, is formed by the printer's configuration driver and transferred to the PrintSpooler service.

For network printers, Windows delivers print jobs on that computer which has a shared printer installed, and only there the PrintSpooler passes print jobs to the print processor driver; so the processing of print jobs can and should be done centrally.

Print processor included in virtual printer in no wise processes received print jobs and serves only for integration with software which includes a virtual printer. The integration is performed only by COM interface declared in Print to Document and implemented by such software. Further on in this documentation a module that implements integration interface will be called Connector.

Print to Document provides few integration modes at connector-module developer's choice. A mode (or connector type) is being set during printer installation: a type of connector-module and GUID of the class that implements the necessary interface are passed to the "InstallPrinterConnector" method (refer to the setup methods).

Connector Types

P2D_CONN_DIRECT

This type provides the closest integration with the print processor. And it also gives to the connector-module's developer the capability of print job data receiving the in short time and in the most effective way to organize the processing. By choosing this mode the developer loses some helpful capabilities from other modes. For example, this mode does not provide a service that extracts EMF for separate pages from the spool data; also using of the host program will be impossible.

The connector-module has to implement IP2DHSJobProcessor70 COM interface. The module can exist as a dll-file or as a program. It should be registered during the printer's installation process in the COM interface standard way.

The developer has to remember that:

- this type of the connector-module works only in the system user context;
- print jobs for the same virtual printer are transferred to the connector-module sequentially, i.e. until the connector completes the processing of one job, the other jobs (if they exist) will be waiting;
- the connector cannot use UI-massages or dialogs for user communication.

P2D_CONN_HOST_SYS

This integration mode uses the host program and all its advantages. In this mode the host program will be launched in the system user context, despite the user which had printed on the virtual printer. Because of that the connector cannot use massages or dialogs for end-user communication.

The connector-module has to implement IP2DConnector70 COM interface. The module can exist as a dll-file or as a program. It should be registered during the printer's installation process in the COM interface standard way.

P2D_CONN_HOST_IAU

This integration mode is similar to the P2D_CONN_HOST_SYS mode.
In this mode the host program will be launched in the context of the active user session (even if several local users logged on), in spite of which user has printed on the virtual printer. If only one user session available, and this session is remote (Remote Desktop Connection), than the host program will be launched in the context of this user. If there is no interactive user session during printing or several remote sessions available, the print job will be skipped without connector notification.

P2D_CONN_HOST_JOU

This integration mode is similar to the P2D_CONN_HOST_SYS mode.
In this mode the host program will be launched in the context of that user which had printed on the virtual printer, if the session (local or remote) of such user is active during the print job processing. In other case the print job will be skipped without connector notification. For print jobs that were printed in the system user context the print job will be skipped also.

Connector Interfaces

IP2DHSJobProcessor70 Interface

This COM interface is used by the printer's print processor driver for communication with the host program and with the connector-module in P2D_CONN_DIRECT mode.

```
HRESULT ProcessJob([in] IP2DJob70* pJob);
```

This method is called by the driver for notification about the necessity of the next print job processing.

pJob - object that implements the IP2DJob70 interface and provides with the information about the printer and the print job.

IP2DJob70 Interface

This COM interface allows the connector-module receiving information about the printer, print job and its data in the P2D_CONN_DIRECT mode.

```
BSTR get_PrinterName();
```

Name of the printer by which the print job was received.

```
BSTR get_DocumentName();
```

Document name (print job) that was entered during printing.

```
ULONG get_JobID();
```

Print job identifier assigned in the PrintSpooler.

```
HRESULT get_JobInfo([in] IStream* pJobInfo);
```

Data block with detailed information about print job (refer to Job Info).

```
HRESULT get_SpoolData([in] IStream* pSpoolData);
```

Data block in universal format (spool EMF format) for the current print job (refer to the "Enhanced Metafile Spool Format Specification" of the Microsoft)

IP2DConnector70 Interface

This COM interface is used for communication between the host program and the connector-module in all modes but P2D_CONN_DIRECT.

```
HRESULT ProcessJob([in] IP2DHSJob70* pJob);
```

This method is called by the print processor for notification about the necessity of the next print job processing.

pJob - object that implements the IP2DHSJob70 interface and provides with the information about the printer and the print job.

```
HRESULT IsAlive();
```

The method is used for verifying of the object status that implements the interface. The method does not have any functionalities but its existence.

IP2DHSJob70 Interface

This COM interface allows the connector-module receiving the information about the printer, print job and its data in any mode but P2D_CONN_DIRECT.

```
BSTR get_PrinterName();
```

Name of the printer by which the print job was received.

```
BSTR get_DocumentName();
```

Document name (print job) that was entered during printing.

```
BSTR get_JobID();
```

Print job identifier assigned by the **Print to Document** configuration driver or by printing application (refer to the Escape code P2D_ESCAPE_SET_JOB_ID).

```
HRESULT get_JobInfo([in] IStream* pJobInfo);
```

Data block with detailed information about print job (refer to Job Info).

```
ULONG get_PageCount();
```

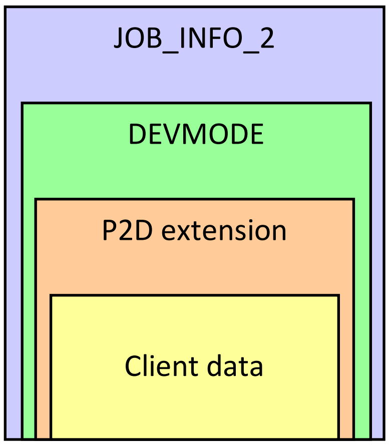
Page quantity of the print job.

```
IStream* get_PageData([in] ULONG PageIndex);
```

Data block in EMF format for the page with the specified index. Can be NULL if the print job does not contain EMF for the specified page. For printer that functions in the trial mode, this method returns EMF with the "Print to Document" text for random pages.

Job Info

Data block associated with the print job that contains detailed information about this print job. Part of the data from this block is formed by the PrintSpooler service; data format corresponds **JOB_INFO_2** structure defined by Windows Platform SDK.



Data block migrates between processes and because of that all structure fields that are documented as pointers should be interpreted not as pointers in the memory but as offsets from the beginning of the block. If the value of the field is a zero offset, this field should be ignored. **Print to Document** driver provides replacing a pointer with an offset. Strings in the block are in Unicode format.

Structure field "pDevMode" contains an offset from the beginning of the block on the data in format that corresponds with DEVMODE structure defined by Windows Platform SDK.

Initially (before printing) this structure is created by the printer’s configuration driver and can be modified by a printing application.

Usually (if "dmDriverExtra" field of DEVMODE has non-zero value), this structure is followed by fields with additional information (P2D extension) provided by the **Print to Document** configuration driver. For print jobs received from a network computer, all the information is about that computer where the printing was performed.

Offset of the P2D extension data relatively to the beginning of the DEVMODE structure is shown in "dmSize" field of this structure.

Job Info (Specific for P2D Printers)

dwSign

Print to Document extension signature "-D2P", 4 bytes.

dwSize

DEVMODE size with additional fields defined by P2D, 4 bytes.

dwFlags

Flag field, provides the information about the printer and Windows version, 4 bytes.

dwLeftMargin

Left margin (none-printed zone) of the printer. The size is shown in 0.1 mm, 4 bytes.

dwTopMargin

Top margin (none-printed zone) of the printer. The size is shown in 0.1 mm, 4 bytes.

dwRightMargin

Right margin (none-printed zone) of the printer. The size is shown in 0.1 mm, 4 bytes.

dwBottomMargin

Bottom margin (none-printed zone) of the printer. The size is shown in 0.1 mm, 4 bytes.

dwProcessID

Identifier of the system process that has printed the data, 4 bytes.

Time

Time when the data have been printed. Size of FILETIME structure.

PrinterID

P2D printer identifier, 40 bytes.

JobID

Identifier that was assigned to the print job by the printing application (refer to Escape code P2D_ESCAPE_SET_JOB_ID) or by the configuration driver. The driver uses GUID as identifier and guaranties its uniqueness. 40 bytes.

lpAppPath

Actual path to the process module that printed data on the printer, MAX_PATH bytes.

lpUserName

User name in which context the process worked, 128 bytes.

lpMachineName

Network name of the computer where the data was printed on the printer, MAX_COMPUTERNAME_LENGTH bytes.

lpHardwareCode

Reserved field, 16 bytes.

Client data

If Job info data block is not limited by the fields above then the next 4 bytes contain the size of product-orientated data. If the size is not zero then it is followed by a data. The maximum size of this data can be 4Kbytes. This data can contain the actual state (specific settings) of the P2D printer which is transferred by the printing application or printer's configuration module to the connector.